МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Огурцов В. В.
Гриньов Д. В.
Щербаков О. В.

ОСНОВИ ВЕБ ТА ВЕБ-ДИЗАЙН, ПРОГРАМУВАННЯ НА БОЦІ КЛІЄНТА

Лабораторний практикум
з навчальної дисципліни
"Веб-технології та веб-дизайн"

Харків. ХНЕУ ім. С. Кузнеця, 2015

УДК 004.738.5(07)
ББК 32.973.202-018.2
Л 12

Рецензенти: докт. техн. наук, ст. наук. співробітник, в. о. завідувача кафедри обчислювальної техніки і програмування Національного технічного університету "ХПІ" *Семенов С. Г.*; докт. техн. наук, професор, провідний науковий співробітник науково-дослідного відділу Харківського університету Повітряних Сил ім. Івана Кожедуба *Кучук Г. А.*

Рекомендовано до видання рішенням вченої ради Харківського національного економічного університету імені Семена Кузнеця.
Протокол № 3 від 29.10.2014 р.

Авторський колектив: канд. екон. наук, доцент Огурцов В. В. – лабораторні роботи 4, 5, 6; канд. техн. наук, доцент Гриньов Д. В. – лабораторна робота 1; канд. техн. наук, професор Щербаков О. В. – лабораторні роботи 2, 3.

Л 12 Основи веб та веб-дизайн, програмування на боці клієнта. Лабораторний практикум з навчальної дисципліни "Веб-технології та веб-дизайн" для студентів напряму підготовки 6.050101 "Комп'ютерні науки" / В. В. Огурцов, Д. В. Гриньов, О. В. Щербаков. – Х. : ХНЕУ ім. С. Кузнеця, 2015. – 208 с. (Укр. мов.)
ISBN 978-966-676-585-0

Призначено для практичного вивчення основ веб-технологій, веб-дизайну та програмування на боці клієнта. Розглянуто основи побудови веб-сайта з використанням HTML, CSS, JavaScript. Досліджено принципи клієнт-серверних технологій. Запропоновано практичні основи створення якісних веб-сайтів.

Рекомендовано для студентів та аспірантів, які спеціалізуються в галузі інформаційних технологій у різних сферах діяльності.

УДК 004.738.5(07)
ББК 32.973.202-018.2

ISBN 978-966-676-585-0

Вступ

Нова інформаційна технологія досягла такого розвитку, що, напевне, не залишилося сфер людського життя, які не задіяні в глобальній мережі Інтернет. Наразі інтерес до мережі Інтернет продовжує зростати. Розроблений у роки інформаційного вибуху, Інтернет стає невід'ємною частиною життя більшості людей усього світу.

Якісний сайт стає важливим, а у деяких галузях – єдиним засобом досягнення економічних, політичних, соціальних, рекламних та інших цілей. Якісний сайт відрізняється від інших сайтів глобальної мережі такими рисами: висока якість інформаційного наповнення й грамотність його подачі; оригінальність і естетична привабливість зовнішнього вигляду сторінок; доступність змісту сайту для максимально широкого кола користувачів поза залежністю від застосовуваних ними типів пристроїв і версій браузерів, а також від обмежень за станом здоров'я; ергономічність елементів користувальницького інтерфейсу сайта, що забезпечує високу ефективність, але водночас легкість і невимушеність взаємодії відвідувача з веб-ресурсом; надійність і безпека використовуваних технологічних рішень, чітка погодженість роботи всіх компонентів; бездоганне пророблення всіх деталей і нюансів.

Звісно, для створення якісного веб-сайта, тобто відповідності до цих рис, потрібна плідна робота висококваліфікованих спеціалістів із різних веб-технологій, які повинні розуміти не лише вузьку галузь знань, але і добре уявляти весь спектр веб-технологій.

У практикумі лабораторні завдання підібрані таким чином, який дозволяє поступово вивчати важливі питання дисципліни з веб-дизайну та програмування на боці клієнта. Поступове навантаження понятійного апарату дасть змогу глибше зрозуміти сутність питань, пов’язаних з механізмами побудови веб-сайтів. Навчальний посібник містить опис шести лабораторних робіт. Кожна з них розкриває відповідну тему із навчальної дисципліни, освітлюючи практичні питання з веб-дизайну та програмування на боці клієнта. Склад кожної лабораторної роботи містить такі пункти:

мета лабораторної роботи;

рекомендації щодо підготовки до виконання роботи;

основні тези;

завдання до лабораторної роботи за варіантами;

хід виконання лабораторної роботи (у разі потреби);

контрольні запитання.

У ході проведення лабораторної роботи студент повинен продемонструвати:

а) знання понятійного апарату й основних принципів побудови веб

б) сайта;

в) розуміння принципів функціонування та використання процесів клієнт-серверних технологій;

г) грамотне використання програмного забезпечення для розроблення веб-сайтів.

Вивчення тем та виконання завдань містить такі етапи:

1. Підготовчий етап (до проведення практичного або лабораторного заняття):

а) отримання відповідного до даних методичних рекомендацій

б) завдання, номера варіанта та вимог від викладача;

в) вивчення теоретичного матеріалу за відповідною темою;

г) розроблення алгоритму виконання завдання.

2. Безпосереднє виконання завдання в аудиторії, в комп'ютерному класі обчислювального центру або самостійно (наприклад, вдома):

а) проходження допуску до лабораторної роботи;

б) установлення (за необхідності) конфігурування програмного забезпечення;

в) відпрацьовування завдання за варіантом;

г) аналіз отриманих результатів.

3. Складання звіту та захист роботи. Звіт про лабораторну роботу повинен містити:

а) титульний аркуш із найменуванням лабораторної роботи (ЛР) і даними виконавця;

б) дату виконання;

в) мету роботи;

г) опис завдання;

д) тексти розроблених HTML-документів та програм;

е) результати роботи й їх аналіз;

ж) висновки про роботу.

Усі матеріали звіту необхідно зброшурувати.

У процесі викладання навчальної дисципліни основна увага приділяється оволодінню студентами професійними компетентностями згідно з Національною рамкою кваліфікацій України.

Лабораторна робота № 1
Розроблення веб-сторінок із використанням мови HTML

1.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення мови розмітки гіпертексту HTML і застосування його в процесі створення сайтів.

1.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити мову розмітки гіпертексту HTML, а також розуміти структуру веб-сайта.

1.3. Загальні тези лабораторної роботи

Слово HTML є скороченням від *HyperText Markup Language* – мови розмітки гіпертексту [7; 8]. HTML-документ переважно є простим текстовим файлом, який містить тільки текст. Тому створювати веб-сторінки можна в будь-якому текстовому редакторі.

Коли веб-сторінка відкривається в браузері, він переглядає код HTML, знаходить спеціальні символи, теги та використовує їх для вставки зображень, зміни вигляду тексту, створення посилань на інші веб-сторінки та ін.

Для позначення тегів використовується символ <тег>. Відомі теги двох видів: парні, які виділяють блок тексту (контейнери) (рис. 1.1), і одинарні. Контейнер вимагає завершального тегу, який позначається </ тег>. Теги нечутливі до регістру, тому записи <B> і <b> еквівалентні.

```
<!DOCTYPE HTML>
        <html>
          <head>
            <meta charset="utf-8">
            <title>Контейнери</title>
          </head>
          <body>
    <p><b>Жирний шрифт</b> у тексті.</p>
          </body>
</html>
```

Рис. 1.1. **Використання парних тегів**

Оскільки одночасно можна використовувати будь-яке розумне поєднання тегів, слід пам'ятати про їх вкладеність (рис. 1.2). Один контейнер повинен знаходитися всередині іншого і не перетинатися (рис. 1.3).

```
<!DOCTYPE HTML >
        <html>
          <head>
            <meta charset="utf-8">
            <title>Вкладення тегів</title>
          </head>
          <body>

            <p><b><i>Напівжирний курсивний текст</i></b></p>

        </body>
</html>
```

Рис. 1.2. **Правильне поєднання тегів**

У даному прикладі текст знаходиться всередині контейнера <I>, який встановлює курсивне накреслення. Він, у свою чергу, розміщується всередині контейнера <B>, що задає жирне накреслення тексту. Результат залишиться незмінним, якщо в даному випадку поміняти теги місцями (рис. 1.3).

```
<!DOCTYPE HTML>
        <html>
          <head>
            <meta charset="utf-8">
            <title>Помилка у вкладенні тегів</title>
          </head>

          <body>

            <p><i><b>Напівжирний курсивний текст</i></b></p>

          </body>
</html>
```

Рис. 1.3. **Неправильне поєднання тегів**

У даному прикладі порушена вкладеність тегів один в іншій. Хоча браузер і відобразить приклад коректно, самостійно "здогадавшись", чого від нього хочуть, подібних помилок слід уникати, оскільки вони призводять до уповільнення роботи сторінки і до неправильної демонстрації сторінки в більшості випадків [10; 12; 14].

Структура документа

Всі нормальні веб-сторінки складаються з двох розділів – заголовка (<HEAD>) і тіла документа (<BODY>). Розділ заголовка може містити текст і теги, але вміст цього розділу не показується безпосередньо на сторінці. Тіло документа призначене для розміщення тегів і змістовної частини (рис. 1.4).

```
<!DOCTYPE HTML >
        <html>
          <head>
        <!-- Цей розділ призначений для заголовка сторінки та технічної інформації. -->
          </head>

          <body>
          <!—А тут треба розміщувати все, що бажано побачити на сторінці. -->
          </body>
</html>
```

Рис. 1.4. **Найпростіший HTML-документ**

DOCTYPE

Елемент <! DOCTYPE> призначений для вказівки на тип поточного документа – DTD (document type definition, опис типу документа). Це необхідно, щоб браузер розумів, як слід інтерпретувати поточну веб-сторінку, адже HTML існує в декількох версіях, крім того є XHTML (EXtensible HyperText Markup Language – розширена мова розмітки гіпертексту), схожий на HTML, але відрізняється від нього за синтаксисом; щоб браузер "не плутався" і розумів, згідно з яким стандартом відображати веб-сторінку та як необхідно в першому рядку коду задавати <! DOCTYPE>.

Існує декілька видів <! DOCTYPE>, вони розрізняються залежно від версії HTML, на яку орієнтовані. У табл. 1.1 наведені основні типи документів з їх описом.

Таблиця 1.1

Допустимі DTD

DOCTYPE	Опис
HTML 5	
<!DOCTYPE HTML>	Синтаксис HTML 5
HTML 4.01	
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	Строгий синтаксис HTML
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	Перехідний синтаксис HTML
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">	У HTML-документі застосовуються фрейми
XHTML 1.0	
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">	Строгий синтаксис XHTML
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">	Перехідний синтаксис XHTML
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">	Документ написаний на XHTML і містить фрейми
XHTML 1.1	
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">	Розробники XHTML 1.1 припускають, що він поступово витісняє HTML

Розділ заголовка документа (<HEAD>)

Теги та тексти, що знаходяться в цьому розділі, не відображуються на веб-сторінці. Цей розділ, як правило, призначений для наступної службової інформації.

Заголовок сторінки (тег <TITLE>)

Використовується для відображення рядка тексту в назві закладки вікна браузера. Такий рядок повідомляє користувачеві назву сайта та іншу інформацію, яку додає розробник.

CSS (Cascading Style Sheets, каскадні таблиці стилів)

Стилі зберігають набір елементів форматування, який застосовується до тексту документа, щоб швидко змінити його зовнішній вигляд.

Метатеги (тег <META>)

Метатеги використовуються для зберігання інформації, призначеної для браузерів і пошукових систем. Наприклад, механізми пошукових систем звертаються до метатегів для отримання опису сайта, ключових слів та інших даних. Хоча тег <META> всього один, він має безліч параметрів.

Два метатеги призначені спеціально для пошукових серверів: *Description* (опис) і *Keywords* (ключові слова).

```
<meta name="Description" content="Сайт про HTML">
```

Більшість пошукових серверів відображують вміст поля *Description* під час виведення результатів пошуку. Якщо цього тегу нема на сторінці, то пошуковий движок просто перерахує перші слова, які зустрічаються на сторінці.

```
<meta name="Keywords" content="HTML, META, метатег">
```

Метатег *Keywords* був призначений для опису ключових слів, що зустрічаються на сторінці. Але в результаті дії користувачів, які бажають потрапити у верхні рядки пошукових систем будь-якими засобами, наразі дискредитований. Тому більшість пошукових систем просто пропускають цей параметр.

Щоб автоматично завантажувати новий документ через визначений проміжок часу, використовується інструкція HTTP-EQUIV = "REFRESH":

```
<meta http-equiv="REFRESH" content="5; URL=http://www.htmlbook.ru">
```

Щоб повідомити браузеру, в якому кодуванні знаходяться символи веб-сторінки, необхідно встановити параметр:

<meta http-equiv="Content-Type" content="text/html; charset=ім'я кодування">.

У стандарті HTML-5 цей рядок може виглядати наступним чином

<meta charset="ім'я кодування">.

Для операційної системи Windows у кирилиці аргумент charset звичайно приймає значення *windows-1251* чи *utf-8*.

Якщо вказівка на кодування відсутня, браузер намагається сам визначити, який тип символів використовується в документі, і вибирає необхідне кодування автоматично. Браузер не завжди може точно розпізнати мову веб-сторінки і в деяких випадках пропонує в'єтнамське кодування замість кирилиці. З цієї причини треба завжди вказувати наведений рядок.

Скрипти

Скриптом традиційно називають програму, яка впроваджується в тіло веб-сторінки та виконує на ній певні дії. Поширеною мовою програмування для написання скриптів є JavaScript.

Порядок тегів в заголовку документа принципового значення не має.

Тіло документа (<BODY>)

Тіло документа призначене для відображення даних на веб-сторінці, зокрема, в тілі розміщується текст, зображення, посилання, таблиці, списки тощо.

Коментарі

Як показано на рис. 1.4, певний текст можна приховати від показу в браузері, зробивши його коментарем. Хоча такий текст користувач не побачить, він буде передаватися в документі, тому, подивившись вихідний код, можна виявити приховану інформацію.

Коментарі починаються тегом <!-- і закінчуються тегом -->. Усе, що знаходиться між цими тегами, відображатися на веб-сторінці не буде.

Робота з текстом

Форматування тексту

Форматування тексту – це засоби його зміни – такі, як вибір накреслення шрифту та використання ефектів, що дозволяють змінювати вид тексту. У табл. 1.2 перелічені основні теги, які застосовуються для зміни оформлення тексту.

Таблиця 1. 2

Теги для форматування тексту

Код HTML	Опис	Приклад
Текст	Жирне накреслення тексту	Текст
Текст	Курсив тексту	Текст
^{Текст}	Верхній індекс	$e=mc^2$
_{Текст}	Нижній індекс	H_2O
<pre>Текст</pre>	Текст пишеться як є, включаючи всі прогалини	Текст
Текст	Акцентування тексту	Текст
Текст	Важливий текст	Текст

Слід зазначити, що теги <B> і <STRONG>, також як <I> і <EM>, є не зовсім еквівалентними та замінними. Перший тег <B> є тегом фізичної розмітки та встановлює жирний текст, а тег <STRONG> – тегом логічної розмітки та визначає важливість поміченого тексту. Такий поділ тегів на логічне та фізичне форматування спочатку призначався для того, щоб зробити HTML універсальним, в тому числі не залежним від пристрою виведення інформації. Теоретично, якщо скористатися, наприклад, мовним браузером, текст, оформлений за допомогою тегів <B> і <STRONG>, буде відзначений по-різному. Однак сталося так, що в популярних браузерах результат використання цих тегів рівнозначний.

Будь-які теги форматування тексту можна використовувати спільно.

Розмір тексту

Для зміни розміру тексту існує кілька можливостей – це використання заголовків <H1>,...,<H6>, тегів <BIG> і <SMALL>. У табл. 1.3 перелічені основні варіанти з описом і прикладом.

Теги <BIG> і <SMALL> можна повторювати кілька разів поспіль, тим самим збільшуючи або зменшуючи текст до потрібних розмірів.

Серед поданих у табл. 1.3 тегів переважно застосовуються теги <H1>, <H2> і <H3>. Вони призначені для створення заголовків до розділів і показують їх відносну важливість. Так, за замовчуванням, текст всередині тегу <H1> відображується в жирному накресленні та розміром 24-х пунктів. Вміст тегу <H2> вже має розмір 18 пунктів, а <H3> – 14 пунктів.

Таблиця 1.3

Теги для зміни розміру тексту

Код HTML	Опис	Приклад
<big>Текст</big>	Збільшує розмір шрифту	Текст
Текст	Зменшує розмір шрифту	Текст
Текст	Пише текст у вигляді великого заголовка	Текст
Текст	Пише текст у вигляді маленького заголовка	Текст

Вирівнювання тексту

Вирівнювання тексту визначає його зовнішній вигляд, орієнтацію країв абзацу та може виконуватися по лівому або правому краю, по центру або по ширині. У табл. 1.4 показані варіанти вирівнювання блоку тексту.

Найбільш поширений варіант – вирівнювання по лівому краю, коли зліва текст зсувається до краю, а правий залишається нерівним. Вирівнювання по правому краю та по центру в основному використовується в заголовках і короткому змісті. Слід мати на увазі, що при використанні вирівнювання по ширині в тексті між словами можуть з'явитися великі інтервали, що не дуже гарно.

Таблиця 1.4

Способи вирівнювання тексту

Вирівнювання по лівому краю	Вирівнювання по правому краю	Вирівнювання по центру	Вирівнювання по ширині
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat	Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat	Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat	Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat

Для установки вирівнювання тексту зазвичай використовується тег параграфа <P> з параметром align, який визначає спосіб вирівнювання. Також блок тексту допустимо вирівнювати за допомогою тегу <DIV> з аналогічним параметром align, як показано в табл. 1.5.

Таблиця 1.5

Вирівнювання тексту за допомогою параметра align

Код HTML	Опис
Текст	Додає новий параграф, за замовчуванням вирівняний по лівому краю. Перед параграфом і після нього автоматично додаються невеликі вертикальні відступи
Текст	Вирівнювання по центру
Текст	Вирівнювання по лівому краю
Текст	Вирівнювання по правому краю
Текст	Вирівнювання по ширині
<nobr>Текст</nobr>	Відключає автоматичне перенесення рядків, навіть якщо текст ширший від вікна браузера
Текст<wbr>	Дозволяє браузеру робити перенесення рядка в зазначеному місці, навіть якщо використовується NOBR тег
Текст	Вирівнювання по центру
Текст	Вирівнювання по лівому краю
Текст	Вирівнювання по правому краю
Текст	Вирівнювання по ширині

Вирівнювання елементів по лівому краю задано за замовчуванням, тому вказувати його ще раз необхідності немає. Тоді параметр *align = "left"* можна опустити.

Відмінність між параграфом (тег <P>) і тегом <DIV> в тому, що на початку й у кінці параграфа з'являється вертикальний відступ, чого немає у випадку використання тегу <DIV>.

Параметр *align* достатньо універсальний і може застосовуватися не тільки до основного тексту, а й до заголовків, наприклад <H1>.

Робота із зображеннями

Вставка зображень. Для вбудовування зображення в документ використовується тег <IMG>, що має обов'язковий параметр *src*, який визначає адресу файла з картинкою і *alt*, що визначає альтернативний текст. Загальний синтаксис для додавання зображення наступний:

<img src="URL" alt="альтернативний текст">

Закривати тег не потрібно, URL (*Universal Resource Locator*, універсальний покажчик ресурсів) – це шлях до графічного файла. Для вказівки на нього можна використовувати як абсолютну, так і відносну адресу.

Доцільно розглянути кілька вказівок на шлях до малюнка для розміщення його на веб-сторінці. Для прикладу обирається файл з малюнком, який називається *sample.gif* і розміщується в папці *images* кореня сайту.

Якщо на початку адреси стоїть слеш (символ /), це означає, що відлік іде від кореня сайту. Наприклад, адреса сайта – *http://htmlbook.ru*. Отже, шлях до зображення */images/bird.jpg*, є вказівкою серверу, що показати слід файл *http://htmlbook.ru/images/bird.jpg*. Слід враховувати, що подібні посилання зі слешем попереду працюють тільки на веб-сервері, на локальному комп'ютері вони діяти не будуть.

Якщо перед адресою додається згадка протоколу *http (http://)*, то мова йде про абсолютне посилання. Зображення завжди буде завантажуватися з зазначеної адреси в Інтернеті, навіть за умови збереження веб-сторінки на локальному комп'ютері.

Двокрапка зі слешем (.. /) на початку адреси говорить про те, що малюнок і веб-сторінка знаходяться в різних папках, які розміщені на одному рівні. На рис. 1.5 (а) показаний файл index.html, в який потрібно помістити малюнок pic.gif. Тоді відносний шлях до зображення з index.html буде*../images/pic.gif*. Можливі випадки зберігання файлів, де звернення з одного файла до іншого перетворюється на набір двокрапок, наприклад:*../../../Images/pic.gif*.

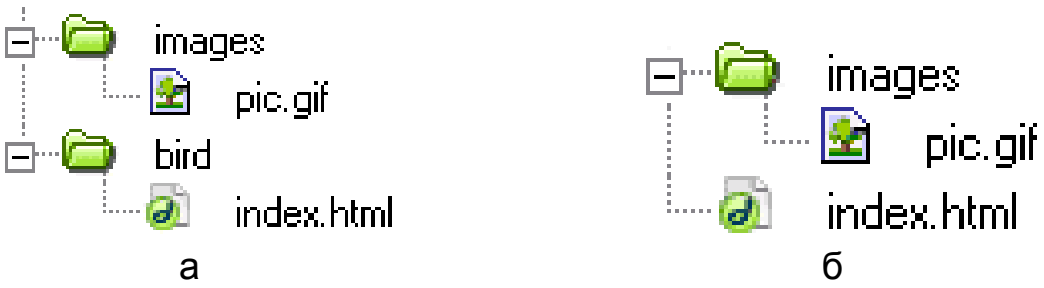


Рис. 1.5. **Приклади розміщення файла**

Ім'я папки на початку шляху, без жодних слешів і двокрапок, повідомляє, що поточний файл і папка з зображенням знаходяться на одному рівні. Як показано на рис. 1.5 (б), відносний шлях до малюнка *pic.gif* з файла *index.html* буде *images/pic.gif*.

На рис. 1.6 показано кілька способів додавання малюнка на веб-сторінку.

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

             <p><img src="http://www.htmlbook/images/sample.gif" alt="Це
абсолютна адреса розміщення зображення"></p>
             <p><img src="/images/sample.gif" alt="Адреса розміщення
зображення щодо кореня сайта"></p>
             <p><img src="images/sample.gif" alt="Адреса розміщення
зображення щодо поточного HTML–документа"></p>

          </body>
</html>
```

Рис. 1.6. **Вставка зображення в документ**

Як правило, в якості формату графічного файла виступає GIF і JPEG.

Формати графічних файлів

Широкого поширення для веб-графіки отримали формати GIF і JPEG. Їх багатофункціональність, універсальність, невеликий обсяг вихідних файлів з достатньою для сайта якістю справили позитивну службу, фактично визначивши їх як стандарт веб-зображень. Є формат PNG, який також підтримується браузерами при додаванні зображень і існує в двох іпостасях – PNG-8 і PNG-24. Проте популярність PNG сильно поступається за визнанням форматам GIF і JPEG.

GIF

GIF (*Graphics Interchange Format*) – формат графічних файлів, широко застосовуваний при створенні сайтів. GIF використовує 8-бітовий колір і ефективно стискає суцільні кольорові області, при цьому зберігаючи деталі зображення.

Особливості GIF

Кількість кольорів у зображенні може бути від 2-х до 256-ти, але це можуть бути будь-які кольори з 24-бітової палітри.

Файл у форматі GIF може містити прозорі ділянки. Якщо використовується відмінний від білого кольору фон, він буде просвічувати крізь "отвори" в зображенні.

Підтримує покадрову зміну зображень, що робить формат популярним для створення банерів і простих анімацій.

Використовує вільний від втрат метод стиснення.

Сфера застосування GIF

Текст, логотипи, ілюстрації з чіткими краями, анімовані малюнки, зображення з прозорими ділянками, банери.

JPEG

JPEG (*Joint Photographic Experts Group*) – популярний формат графічних файлів, широко використовуваний у процессі створення сайтів і зберігання зображень. JPEG підтримує 24-бітовий колір і зберігає яскравість і відтінки кольорів у фотографіях незмінними. Даний формат називають стисненням зі втратами, оскільки алгоритм JPEG вибірково відкидає дані. Метод стиснення може внести спотворення в малюнок, особливо якщо він містить текст, дрібні деталі або чіткі краї. Формат JPEG не підтримує прозорість. Коли зберігається фотографія в цьому форматі, прозорі пікселі заповнюються певним кольором.

Особливості JPEG

Кількість кольорів у зображенні – близько 16-ти мільйонів, чого цілком достатньо для збереження фотографічної якості зображення.

Основна характеристика формату – якість, що дозволяє управляти кінцевим розміром файла.

Підтримує технологію, так званий прогресивний JPEG, в якому версія малюнка з низькою здатністю з'являється у вікні перегляду до повного завантаження самого зображення.

Сфера застосування JPEG

Використовується переважно для фотографій. Недоцільно використовувати для малюнків, які містять прозорі ділянки, дрібні деталі або текст.

PNG-8

PNG-8 (*Portable Network Graphics*) – формат, за своїми діями аналогічний GIF. За запевненням розробників, використовує покращений формат стиснення даних, але, як показує практика, це не завжди так.

Особливості PNG-8

Використовує 8-бітову палітру (256 кольорів) у зображенні, за що й отримав у своїй назві цифру вісім. При цьому можна вибирати, скільки кольорів буде зберігатися у файлі: від 2-х до 256-ти.

На відміну від GIF, не відображує анімацію в жодному вигляді.

Сфера застосування PNG-8

Текст, логотипи, ілюстрації з чіткими краями, зображення з градієнтною прозорістю.

PNG-24

PNG-24 – формат, аналогічний до PNG-8, але використовує 24-бітову палітру кольорів. Подібно до формату JPEG, зберігає яскравість і відтінки кольорів у фотографіях. Подібно до GIF і формату PNG-8, зберігає деталі зображення, як, наприклад, в лінійних малюнках, логотипах або ілюстраціях.

Особливості PNG-24

Використовує приблизно 16,7 млн кольорів у файлі, через що цей формат застосовується для повнокольорових зображень.

Підтримує багаторівневу прозорість, що дозволяє створювати плавний перехід від прозорої області зображення до кольорової – так званий градієнт.

Через те, що використовуваний алгоритм стиснення зберігає всі кольори та пікселі в зображенні незмінними порівняно з іншими форматами, у PNG-24 кінцевий обсяг графічного файла є найбільшим.

Сфера застосування PNG-24

Фотографії, малюнки, що містять прозорі ділянки, малюнки з великою кількістю кольорів і чіткими краями зображень.

Розміри зображення

Для зміни розмірів зображення засобами HTML передбачені параметри *width* і *height* тегу <IMG>.

Початкове зображення *sample.gif* має розміри 100 x 111 пікселів (рис. 1.7).

	**<img** src="sample.gif" width="100" height="111" alt="початковий розмір"**>**

Рис. 1.7. **Вставка малюнка вихідного розміру за допомогою HTML**

Необхідно обов'язково задавати розміри всіх зображень на веб-сторінці. Це дещо прискорює завантаження сторінки, оскільки браузеру нема потреби обчислювати розмір кожного малюнка після його отримання. Це твердження особливо важливе для зображень, розміщених усередині таблиці. Така таблиця не буде висвітлена до тих пір, поки всі малюнки в ній не будуть завантажені повністю.

Ширину та висоту зображення можна змінювати як в меншу, так і в більшу сторону. Однак на швидкість завантаження малюнка це не впливає, оскільки розмір файлу залишається незмінним. Треба з обережністю зменшувати зображення, тому що це може викликати подив

у користувачів: чому такий маленький малюнок так довго завантажується? Збільшення ж розмірів призводить до зворотного ефекту – розмір зображення великий, але файл щодо зображення аналогічного розміру завантажується швидше.

Нижче наведено малюнок *sample.gif* зі збільшеною вдвічі шириною та висотою (200 x 222) (рис. 1.8).

	**<img** src="sample.gif" width="200" height="222" alt='Збільшений розмір"**>**

Рис. 1.8. **Вставка малюнка збільшеного розміру**

Алгоритм, що використовується браузерами для ресемплірування (зміни розміру) зображення, поступається за своїми можливостями графічним редакторам. Тому збільшувати зображення шляхом зміни їх розмірів потрібно в особливих випадках, надто якщо погіршується якість малюнка. Краще скористатися будь-якою графічною програмою. Винятком є малюнки, які містять прямокутні області.

Зображення, яке додається на веб-сторінку, можна помістити в рамку різної ширини. Для цього служить параметр *border* тегу <IMG>. За замовчуванням, рамка навколо зображення не відображується, за винятком випадку, коли малюнок є посиланням. Колір рамки в цьому випадку збігається з кольором тексту, заданому за допомогою стилю або параметра *text* тегу <BODY> (рис. 1.9).

Якщо зображення є посиланням, рамка додається автоматично, товщина її – один піксель, а колір рамки збігається з кольором посилань.

Щоб прибрати рамку, слід задати параметр *border = "0"* у тегу <IMG>.

Альтернативний текст дозволяє отримати текстову інформацію про малюнок при відключеному в браузері завантаженні зображень. Оскільки завантаження зображень відбувається після отримання браузером інформації про нього, то замісник малюнка – текст з'являється раніше. А вже в міру завантаження текст заміняється зображенням.

```
<!DOCTYPE HTML>
        <html>
          <head>
            <meta charset="utf-8">
            <title>Рамка та малюнок</title>
          </head>
          <body text="#ff0000">
            <p><img src="sample.gif" width="100" height="111"
border="2" alt="Рамка зеленого кольору товщиною 2
пікселя"></p>
          </body>
</html>
```

Рис. 1.9. **Установка рамки для малюнка за допомогою HTML**

Деякі браузери, зокрема *Internet Explorer*, також відображують альтернативний текст у вигляді підказки, що з'являється при наведенні курсору миші на зображення.

Для створення альтернативного тексту використовується параметр *alt* тегу <IMG>.

Для зображень можна вказувати їх розташування щодо тексту або інших зображень на веб-сторінці. Спосіб вирівнювання зображень задається параметром *align* тегу <IMG>. Цей параметр може приймати такі значення: *bottom* (нижня межа зображення вирівнюється за базовою лінією текстового рядка; це значення встановлено за замовчуванням), *left* (зображення розташовується по лівому краю батьківського елементу), *middle* (середина зображення вирівнюється за базовою лінією поточного рядка тексту), *right* (зображення вирівнюється по правому краю батьківського елементу), *top* (верхня межа зображення вирівнюється за найвищим елементом поточного рядка).

Найбільш популярні параметри *left* і *right*, оскільки вони не тільки вирівнюють зображення по краю вікна браузера, а й задають обтікання тексту навколо зображення. Щоб текст не прилягав щільно до малюнка, рекомендується в тезі <IMG> додати параметр *hspace* і *vspace*, які задають відстань до тексту у пікселях.

Посилання

Для створення посилання необхідно повідомити браузеру, що є посиланням, а також вказати адресу документа, на який слід зробити посилання. Обидві дії виконуються за допомогою тегу <A>, який має єдиний обов'язковий параметр *href*. Як значення використовується адреса документа (*URL*).

Адреса посилання може бути абсолютною та відносною. Абсолютні адреси працюють скрізь і всюди незалежно від імені сайту або веб-сторінки, де прописано посилання. Починаються вони зі вказівки протоколу передачі даних. Так, для веб-сторінок це зазвичай *HTTP* (*HyperText Transfer Protocol*, протокол передачі гіпертексту), відповідно, абсолютні посилання починаються з ключового слова *http://* (рис. 1.10).

```
<!DOCTYPE HTML">
        <html>
          <head>
             <meta charset="utf-8">
             <title>Абсолютне посилання</title>
          </head>
          <body>
        <p>
        <a href="http://google.com">Пошукова система Google</a>
        </p>
          </body>
</html>
```

Рис. 1.10. **Використання абсолютних посилань**

Відносні посилання побудовані щодо поточного документа або адреси. Приклади відносних адрес: / або */demo/.*

Ці два посилання називаються неповними і наказують веб-серверу завантажувати файл *index.html* (або *default.html*), який знаходиться в корені сайту або папці *demo*. Якщо файл index.html відсутній, браузер, як правило, показує список файлів, що знаходяться в даному каталозі.

/images/pic.gif

Слеш перед адресою говорить про те, що адресація починається від кореня сайту. Посилання веде на малюнок pic.gif, який знаходиться в папці images. А та, в свою чергу, розміщена в корені сайту.

../help/me.html

Двокрапка перед іменом вказує браузеру перейти на рівень вище в списку каталогів сайту.

manual/info.html

Якщо перед іменом папки нема додаткових символів на зразок двокрапки, то вона розміщена всередині поточного каталогу (рис. 1.11).

```
<!DOCTYPE HTML>
        <html>
          <head>
             <meta charset="utf-8">
             <title>Відносне посилання</title>
          </head>
          <body>

             <p><a href="images/xxx.jpg">Подивіться на моє фото!</a></p>
        <p><a href="tip.html">Як зробити таке фото?</a>
</p>

          </body>
</html>
```

Рис. 1.11. **Використання відносних посилань**

Іноді можна зустріти в адресі посилання шлях у вигляді */file/doc.html*. Точка зі слешем (символ /) означає, що відлік ведеться від поточної папки. Подібний запис зайвий, і його можна скоротити до *file/doc.html*.

Кольори посилань задаються в якості параметрів тегу <BODY>. Параметри є необов'язковими і, якщо вони не вказані, використовуються значення за замовчуванням.

LINK – визначає колір посилань на веб-сторінці. Колір за замовчуванням, синій, # 0000FF.

ALINK – колір активного посилання. Колір посилання змінюється при натисканні на кнопки миші. Колір, за замовчуванням, червоний, # FF0000.

VLINK – колір вже відвіданих посилань. Колір, за замовчуванням фіолетовий, # 800080.

За замовчуванням, при переході за посиланням документ відкривається в поточному вікні або фреймі. За необхідністю, ця умова може бути замінена параметром target тегу <A>. Цей параметр може набувати таких значень:

_blank – завантажує сторінку в нове вікно браузера;

_self – завантажує сторінку в поточне вікно;

_parent – завантажує сторінку у фрейм-батьківську;

_top – скасовує всі фрейми та завантажує сторінку в повному вікні браузера.

Для створення валідного коду параметр *target* може використовуватися тільки при перехідному !DOCTYPE.

Якщо на сайті використовуються фрейми, то як значення *target* можна використовувати імено фрейма.

Якщо у параметрі *target* вказано невідоме значення (наприклад, імено фрейма набрано з помилкою), то це призводить до того, що посилання відкривається в новому вікні.

Якщо на веб-сторінці необхідно зробити, щоб всі посилання відкривалися в новому вікні, то немає необхідності додавати в усі теги <A> параметр *target = "_blank"*.

Засобами тегів HTML прибрати підкреслення у посилань не можна. Тому для цієї мети використовуються каскадні таблиці стилів (*Cascading Style Sheets, CSS*), які будуть розглянуті в лабораторній роботі № 2.

Крім тексту, як посилання можна використовувати малюнки. Зображення в цьому випадку треба помістити між тегами *<a href=...>* і *</ a>*. Параметр *href* тегу <A> задає шлях до документа, на який вказує посилання, а *src* тегу <IMG> – шлях до графічного файла.

Навколо зображення-посилання автоматично додається рамка товщиною в один піксель і кольором, що збігається з кольором текстових посилань. Щоб прибрати рамку, слід у тегу <IMG> встановити параметр *border = "0"*.

Великі документи читаються краще, якщо вони мають зміст з посиланнями на відповідні розділи. Для створення посилання слід спочатку зробити закладку (якір) у відповідному місці та найменувати її за допомогою параметра *name* тегу <A>, як показано на рис. 1.12.

Між тегами *<a name="top">* і *</ a>* відсутній текст, оскільки потрібно лише вказати місце розташування переходу за посиланням, що знаходиться внизу сторінки. Імено посилання на закладку починається

символом #, після чого йде назва закладки. Назва вибирається будь-яка, відповідно до тематики.

```
<!DOCTYPE HTML>
        <html>
          <head>
            <meta charset="utf-8">
            <title>Закладка</title>
          </head>
          <body>
            <p><a name="top"></a></p>
            <p> Тут розташовується текст для прикладу
    </p>
            <p><a href="#top">Наверх</a></p>
          </body>
</html>
```

Рис. 1.12. **Створення внутрішнього посилання**

Можна також робити посилання на закладку, що знаходиться на іншій веб-сторінці і навіть на іншому сайті. Для цього в адресі посилання належить вказати її адресу і в кінці додати символ грат # та імено закладки.

Посилання на адресу електронної пошти здіснюється майже так, як і посилання на веб-сторінку. Тільки замість *http-схеми URL* використовується *mailto-схема* (рис. 1.13). Таке посилання нічим не відрізняється від посилання на веб-сторінку, але при натисканні на неї запускається поштова програма, встановлена за замовчуванням.

```
<!DOCTYPE HTML>
        <html>
          <head>
            <meta charset="utf-8">
            <title>Адреса електронної пошти</title>
          </head>
          <body>
            <p><a href="mailto:ask@site.ua"> Задавайте питання по
електронній пошті </a></p>
        <p><a href="mailto:ask@site.ua?subject=
Питання по HTML">Задавайте питання по електронній пошті</a></p>
          </body>
</html>
```

Рис. 1.13. **Посилання на адресу електронної пошти**

Можна також автоматично додати тему повідомлення, приєднавши до адреси електронної пошти через символ питання – параметр *subject = "тема повідомлення" (*див. рис. 1.13).

Робота зі списками

Марковані списки

Для установки маркованого списку використовуються теги <UL> і <LI> (рис. 1.14).

Що треба враховувати під час тестування сайту: працездатність усіх посилань підтримку різних браузерів читабельність тексту	<html> <head> <title>Список</title> </head> <body> ** **Під час тестування сайту:** ** ** ** ** **працездатність усіх посилань** ** ** **підтримка різних браузерів** ** ** **читабельність тексту** ** ** ** </body> </html>

Рис. 1. 14. **Створення маркованого списку**

Марковані списки дозволяють розбити великий текст на окремі блоки. Тим самим залучається увага читача до тексту та підвищується його читабельність. Із урахуванням того, що сприйняття тексту з екрану монітора важче, ніж з його друкованого варіанта, це є досить корисним прийомом.

Маркери можуть приймати один з трьох видів: коло (за замовчуванням), коло та квадрат. Для вибору типу маркера використовується параметр type = "..." тегу <UL>. Замість трьох крапок підставляється одне з трьох значень, зазначених у табл. 1.6.

Таблиця 1.6

Види маркерів

Код HTML	Приклад
<ul type="disc">	Що слід враховувати в процесі тестування сайта: • працездатність усіх посилань; • підтримку різних браузерів; • читабельність тексту
<ul type="circle">	Що слід враховувати при тестуванні сайта: • працездатність усіх посилань; • підтримку різних браузерів; • читабельність тексту
<ul type="square">	Що слід враховувати при тестуванні сайта: • працездатність всіх посилань; • підтримку різних браузерів; • читабельність тексту

Нумеровані списки

Нумеровані списки є набором елементів з їх порядковими номерами. Вид і тип нумерації залежить від параметрів тегу <OL>, який використовується для створення списку. В якості маркерів можуть бути такі значення:

арабські цифри;

великі латинські літери;

малі латинські літери;

великі римські цифри;

малі римські цифри.

У табл. 1.7 наведені різноманітні параметри тегу <OL> і результат їх застосування.

До і після списку автоматично додаються вертикальні відступи, це є особливістю тегу <OL>.

Таблиця 1.7

Варіанти нумерованих списків

Код HTML	Приклад
  - текст - текст - текст 	Нумерований список із параметрами за замовчуванням: 1. текст; 2. текст; 3. текст
<ol start="5">	Нумерований список, який починається з п’яти: 5. текст; 6. текст; 7. текст
<ol type="A">	Нумерований список із великими літерами латинського алфавіту: A. текст; B. текст; C. текст
<ol type="a’>	Нумерований список із малими літерами латинського алфавіту: a. текст; b. текст; c. текст
<ol type="I’>	Нумерований список із великими римськими цифрами: I. текст; II. текст; III. текст
<ol type="I">	Нумерований список із малими римськими цифрами: i. текст; ii. текст; iii. текст
<ol type="1">	Нумерований список із арабськими цифрами: 1. текст; 2. текст; 3. текст
<ol type="I" start="7">	Список с римськими цифрами, починаючи з семи: VII. текст; VIII. текст; IX. текст

Список визначень є текстом, що складається з двох взаємопов'язаних наборів – списку термінів і списку визначень термінів. Спочатку вказується перший термін, нижче від нього йде його визначення, далі – наступний термін із визначенням тощо.

Структура списку визначень наступна:

Термін 1.

Визначення терміна 1.

Термін 2.

Визначення терміна 2.

Сам список задається за допомогою контейнера <DL>, термін – тегом <DT>, а його визначення – за допомогою тегу <DD>.

Для організації складної ієрархічної структури тексту призначені вкладені списки. Оскільки на веб-сторінці не можна автоматично вести багаторівневу нумерацію (на зразок використання підпунктів типу 1.1 або 2.1.3), доводиться вводити числа самостійно або спрощувати відображення списку.

Таблиці

Таблиця складається з рядків і стовпців осередків, які можуть містити текст і малюнки. Зазвичай таблиці використовуються для впорядкування та подання даних, однак можливості таблиць цим не обмежуються. За допомогою таблиць зручно верстати макети сторінок, розташувавши потрібним чином фрагменти тексту та зображень.

Для додавання таблиці на веб-сторінку використовується тег-контейнер <TABLE>. Таблиця повинна містити хоча б один рядок і колонку (рис. 1.15).

Для додавання рядків використовується тег <TR>. Щоб розділити рядки на колонки, застосовуються теги <TD> і <TH> (рис. 1.16).

```
<!DOCTYPE HTML>
        <html>
          <head>
             <meta charset="utf-8">
             <title>Таблиця</title>
          </head>
          <body>
             <table>
                <tr>
                   <td> Вміст таблиці </td>
                </tr>
             </table>
          </body>
</html>
```

Рис. 1.15. **Створення найпростішої таблиці**

Відмінність між цими тегами у наступному. Тег <TH> призначений для створення заголовків, вміст такої комірки позначається жирним накресленням і вирівнюється по центру (рис. 1.16). В іншому випадку діють ці теги однаково.

Осередок 1	Осередок 2
Осередок 3	Осередок 4

```
<!DOCTYPE HTML>
        <html>
          <head>
             <meta charset="utf-8">
             <title>Таблиця</title>
          </head>
          <body>
             <table border="1">
                <tr>
                   <th>Осередок 1</th>
                   <td>Осередок 2</td>
                </tr>
                <tr>
                   <th>Осередок 3</th>
                   <td>Осередок 4</td>
                </tr>
             </table>
          </body>
</html>
```

Рис. 1.16. **Таблиця з осередками різних типів**

Особливості таблиць

У кожного параметра таблиці є своє значення, встановлене за замовчуванням. Це означає, що якщо якийсь атрибут пропущений, то неявно він все одно присутній, причому з певним значенням, через що вигляд таблиці може виявитися інакшим, ніж припускав розробник. Щоб розуміти, чого можна чекати від таблиць, слід знати їх явні та неявні особливості, зазначені в подальшому тексті.

Одну таблицю допускається поміщати всередину осередків іншої таблиці. Це потрібно для подання складних даних або в тому випадку, коли одна таблиця виступає в ролі модульної сітки, а друга, всередині неї – вже як звичайна таблиця.

Розміри таблиці спочатку не встановлюються й обчислюються на основі вмісту осередків. Наприклад, загальна ширина визначається автоматично, виходячи з сумарної ширини вмісту осередків плюс ширина кордонів між осередками, поля навколо вмісту, що встановлюється через параметр *cellpadding* і відстань між осередками, які визначаються значенням *cellspacing*.

Якщо для таблиці задана її ширина у відсотках або пікселях, то вміст таблиці підлаштовується під зазначені розміри. Так, браузер автоматично додає переноси рядків до тексту, щоб текст повністю помістився в осередок і при цьому ширина таблиці залишилася без змін. Буває, що ширину вмісту комірки неможливо змінити, як це, наприклад, відбувається при додаванні малюнків до осередку. У цьому випадку ширина таблиці збільшується, незважаючи на зазначені розміри.

Рамка таблиці, у разі додавання параметра *border* до тегу <TABLE>, спочатку відображується як тривимірна. Приєднання параметра *bordercolor* перетворює рамку на однотонну, ліквідовуючи тим самим ефект тривимірності.

Поки таблиця не завантажиться повністю, її вміст не почне відображуватися. Справа в тому, що браузер, перш ніж показати вміст таблиці, повинен обчислити необхідні розміри осередків, їх ширину та висоту. А для цього необхідно знати, що в цих осередках знаходиться. Тому браузер і чекає, поки завантажиться все, що знаходиться в осередках, і тільки потім відображує таблицю.

Таблиця, якщо не вказано особливо, завжди вирівнюється по лівому краю. За замовчуванням, таблиця виводиться без рамки. Якщо ширина таблиці не вказана, вона підганяється під зміст осередків.

Можлива ситуація, коли між осередків таблиці виникають зайві порожні проміжки. Це пов'язано з тим, що перенесення рядків у коді HTML автоматично створює і додатковий пробіл у таблиці. Щоб позбутися цього, треба помістити код всередині тегу <TR> в один рядок.

Для зміни вигляду та властивостей таблиці використовується безліч параметрів, які додаються до тегу <TABLE>. Загальний синтаксис наступний:

<table параметр1="..." параметр2="...">

Параметри, які використовуються тільки для тегів <TH> і <TD>, наведено в табл. 1.8.

Таблиця 1.8

Властивості осередків таблиці

Властивість	Значення	Опис	Приклад
nowrap		Забороняє переноси рядків у тексті	<td nowrap>
colspan	n	Кількість об'єднуваних колонок	<td colspan="3">
rowspan	n	Кількість об'єднуваних рядків	<td rowspan="3">

Опис параметрів таблиці та їх значень наведено в табл. 1.9.

Таблиця 1.9

Параметри тегу <TABLE>

Властивість	Значення	Опис	Приклад
1	2	3	4
align	left right center	Вирівнювання таблиці	<table align="center">
background	URL	Визначає зображення, яке буде використовуватися в якості фонового малюнка таблиці	<table background="pic.gif">
bgcolor	#rrggbb	Колір фону таблиці	<table bgcolor="#ff9900">
border	n	Товщина рамки в пікселях	<table border="2">

Закінчення табл. 1.9

1	2	3	4
cellpadding	n	Відстань між осередком і його вмістом	<table cellpadding="7">
cellspacing	n	Дистанція між осередками	<table cellspacing="3">
cols	n	Задає кількість стовпців у таблиці, допомагаючи браузеру в підготовці до її відображення	<table cols="3">
nowrap		Забороняє переноси рядків у тексті	<table nowrap>
frame	void above below lhs rhs hsides vsides box	Задавання типу рамки таблиці	<table frame="hsides">
rules	all groups cols none rows	Визначає, де малювати між осередками	<table rules="cols">
width	n n%	Мінімальна ширина таблиці, можна задавати в пікселях або відсотках	<table width="90%">

Завдяки великій кількості властивостей осередків, можна змінювати вигляд і оформлення таблиць.

У табл. 1.10 наведені параметри, які можуть бути додані до тегів <TR>, <TH> і <TD>.

Таблиця 1.10

Параметри осередків таблиць

Властивість	Значення	Опис	Приклад
align	left right center	Вирівнювання вмісту комірки	<td align="center">
background	URL	Встановлює фоновий малюнок у комірці	<td background="pic.gif">
bgcolor	#rrggbb	Колір фону комірки	<td bgcolor="#ff9900">
valign	top midlle bottom	Вирівнювання вмісту комірки по висоті	<td valign="top">
width	n n%	Мінімальна ширина осередку, можна задавати в пікселях або відсотках	<td width="90%">
height	n n%	Мінімальна висота комірки, можна задавати в пікселях або відсотках	<td height="37">

Вміст комірок за замовчуванням вирівнюється по лівому краю по горизонталі і по центру – по вертикалі.

Параметри тегу <TD> мають більший пріоритет, ніж параметри тегу <TR>, а властивості осередків вищі за властивості самої таблиці.

Браузер Internet Explorer може не застосовувати деякі параметри, прикладені до тегу <TR>. У цьому випадку треба використовувати ті ж аргументи, але до тегу <TD> або <TH>.

Валідація документів

Валідацією називають перевірку документа на відповідність до веб-стандартів і виявлення існуючих помилок. Відповідно, валідним є такий веб-документ, який пройшов подібну процедуру та не має зауважень за кодом. Код веб-сторінки повинен підкорятися певним правилам, які називаються специфікацією, яку розробляє W3 Консорціум (www.w3c.org) за підтримки розробників браузерів.

Валідація документів призначена не тільки для того, щоб упевнитися, що код відповідає специфікації HTML, але і з метою усунення наявних помилок і зауважень в документі. Водночас формування певної культури написання коду дозволяє істотно знизити кількість або навіть взагалі позбутися можливих помилок. Така культура складається зі знання специфікацій і типових "ляпів" розробників, яких треба уникати.

За адресою http://www.w3.org/TR/html401/ ознайомитися з правилами HTML версії 4.01 може кожен, тут же розглядаються рядові помилки, щоб навчити їх обходити.

Помилки в коді зазвичай виникають з наступних причин:

на сторінці не заданий !DOCTYPE;

помилка (невірно написаний тег або його атрибут);

не вказаний обов'язковий параметр тегу;

використовується тег або його параметр, який не входить до специфікації;

неправильне вкладення тегів.

Ці помилки слід розібрати докладніше.

Не вказаний !DOCTYPE

Елемент !DOCTYPE розташовується в першому рядку коду документа та повідомляє браузеру, як інтерпретувати код і відображувати дану веб-сторінку. Відмінність між сторінкою з !DOCTYPE і без нього може бути дуже суттєвою, до того ж валідатор у першу чергу перевіряє наявність цього елемента в коді.

Помилка

Очевидно, що найпростіша для виправлення помилка виникає через описки при друкуванні, коли допущено неправильне написання необхідного тегу. Після валідації видається тип помилки і номер рядка в коді, де вона з’явилась, тому залишається тільки поміняти значення на коректне.

Не вказаний обов'язковий параметр тегу

У деяких тегів є параметри, які обов'язково повинні бути присутніми. Наприклад, не можна просто вказати тег <style>, необхідно писати: <*style type="text/css"*>.

Параметр або значення не входить до специфікації

У прагненні до завоювання ринку користувачів розробники браузерів додавали до них спеціальні теги, що не входять до специфікації HTML, але розширюють можливості веб-сторінок. Із часом частина таких тегів була включена до специфікації, але більшість опинилася "за бортом". При цьому підтримка браузером залишилась, тому результат роботи тегу спостерігати можна, але валідацію документ не пройде. Типовим прикладом подібного тегу є <MARQUEE>, розробленний компанією Microsoft і зрозумілий усім сучасним браузерам. Але до специфікації цей тег не включено.

Невірне вкладення тегів

Помилка зі вкладенням одного контейнера всередину іншого може бути викликана наступними причинами:

блоковий елемент розташовується усередині вбудованого, коли має бути навпаки – вбудовані елементи допустимо поміщати всередину блокових;

перетин тегів, наприклад: текст </ strong> </ em>. Тут завершальний тег </ strong> поміщається в контейнер , тоді як він повинен слідувати тільки після тегу </ em>;

не дотримується порядок вкладення тегів. Для певних елементів зразок списку та таблиці принципового значення має порядок проходження тегів. Перестановка тегів місцями може призвести до неправильного відображення об'єкта та появи помилок при валідації документа.

Наостанок слід нагадати прості правила написання коду, дотримання яких допоможе істотно скоротити кількість помилок або уникнути їх.

1. Закривати всі теги.

Хоча HTML і не вимагає присутності деяких завершальних тегів, їх наявність допоможе зберегти строгість коду та чітко визначити порядок проходження тегів.

2. Вказувати значення параметрів тегів у лапках.

Валідатор у багатьох випадках пропускає значення параметрів, зазначених без жодних лапок, тому лапки краще писати завжди. По-

перше, подібна навичка допоможе усуненню можливих помилок, пов'язаних з параметрами тегів. А по-друге, допоможе легше перейти на XHTML (Extensible Hypertext Markup Language, розширювана мова розмітки гіпертексту) – синтаксично більш сувору версію HTML. У XHTML лапки виступають обов'язковим елементом синтаксису.

3. Колекціонувати заготовки.

Більшість елементів веб-сторінки достатньо шаблонна, тому маючи в своєму запасі набір перевірених заготовок на різні випадки, можна скоротити витрати часу та бути впевненим, що код коректний.

4. Використовувати блокові елементи.

Не можна просто вставити текст в код документа, він повинен розташовуватися усередині абзацу (тег <P>) або іншого блочного елемента. У тих випадках, коли невідомо, який блоковий тег використовувати, треба додавати універсальний елемент <DIV>.

5. Перемикати !DOCTYPE.

У HTML-коді зазвичай застосовується строгий !DOCTYPE, який найбільш повно відповідає специфікації. Однак вимагає дотримання всіх найжорсткіших правил написання коду. У тих випадках, коли це складно або затратно за часом, необхідно перемикати на перехідний !DOCTYPE.

Для перевірки веб-сторінок на наявність помилок і зауважень існує безліч шляхів і способів. Умовно вони поділяються на онлайнові і локальні. Онлайнові призначені для перевірки сторінок за допомогою браузера через Інтернет, а локальні використовуються для перевірки документів на поточному комп'ютері.

За адресою http://validator.w3.org розташовується, мабуть, найпоширеніший з інструментів для перевірки окремих сторінок на валідність. Цей сайт пропонує три способи перевірки: за адресою, локального файлу та введеного до форми коду.

Популярність браузера Firefox обумовлена наявністю в ньому великої кількості різноманітних розширень – програм, які додають нові можливості до браузера. Розширення побудовані за відкритою технологією, написати їх може будь-який розробник. Не залишені без уваги і веб-розробники – для їх зручності створено безліч розширень, у тому числі і для валідації документа прямо в браузері. У даному випадку

цікавий HTML Validator. Ця програма побудована за тією ж технологією, що і валідатор W3C, але не вимагає підключення до Інтернету і працює прямо "на льоту".

При відкритті веб-сторінки HTML Validator одразу починає свою роботу, і результат перевірки відображується в рядку стану, в його правому нижньому кутку у вигляді невеликого малюнка. Зображення, показане на рис. 1.17, залежить від статусу перевірки.

а б в

Рис. 1.17. **Види малюнків, які відображуються під час перевірки документа**

Кільце з галочкою (див. рис. 1.17-а) показує, що документ пройшов стандартизацію, жовтий трикутник зі знаком оклику див. (див. рис. 1.17-б) – за кодом має зауваження, що можуть бути виправлені автоматично. А червоний гурток з хрестиком див. (див. рис. 1.17-в) попереджує про серйозні помилки.

Переглянути всі помилки можна двояко. По-перше, зазирнути в HTML-код документа через меню Вид>, вихідний код сторінки або натиснути правою кнопкою і в контекстному меню вибрати "Перегляд вихідного коду сторінки".

Вікно вихідного коду веб-сторінки розділене на три частини (рис. 1.18), де верхній блок містить власне HTML-код. У лівому нижньому блоці відображується список помилок і зауважень чи інформаційні повідомлення щодо валідного документа. Правий нижній блок призначений для докладних підказок про поточні зауваження.

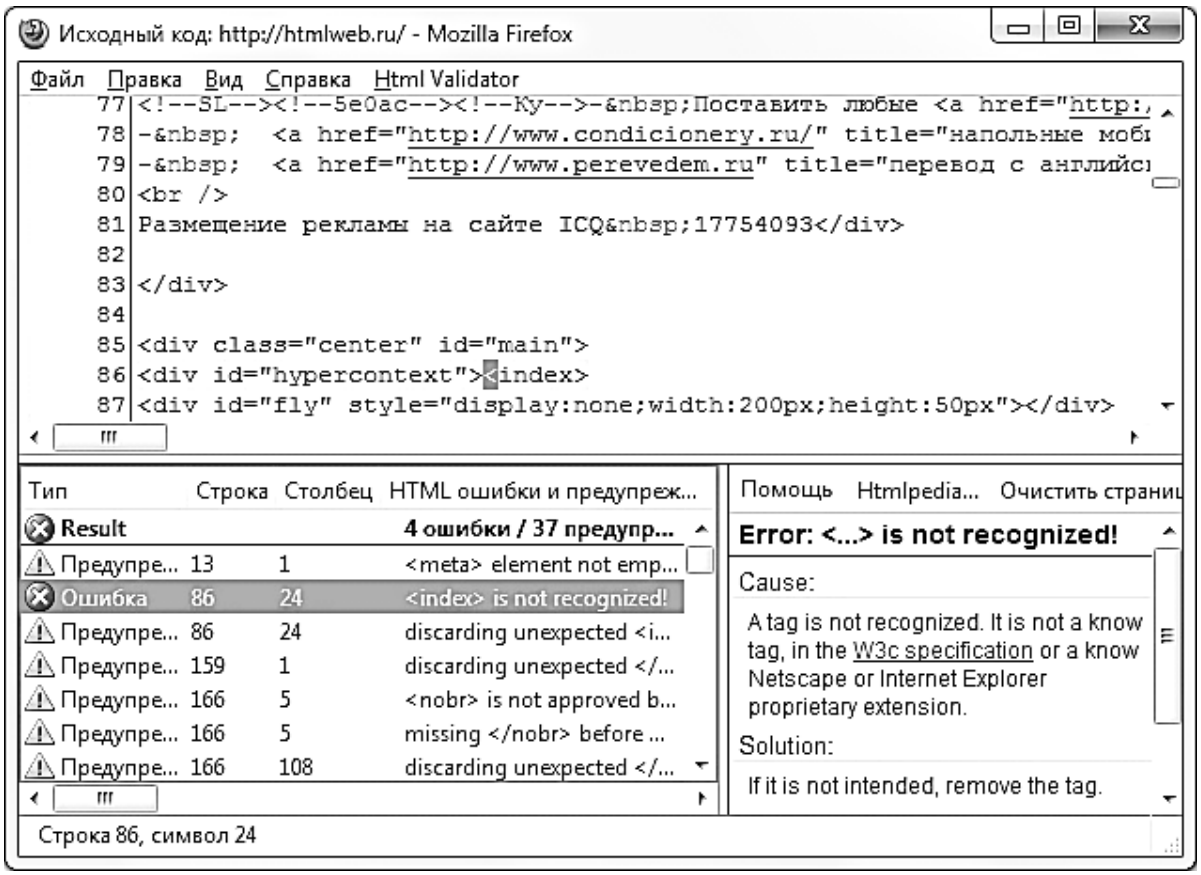


Рис. 1.18. **Результат роботи розширення HTML Validator**

1.4. Завдання до лабораторної роботи

1. Ознайомитися з подібними сайтами, вивчити їх структуру.

2. Продумати та розробити структуру свого сайту: кількість сторінок, їх змістовне наповнення; розміщення текстової, табличної та графічної інформації на кожній сторінці; порядок навігації по сайту.

3. Файл першої (головної) сторінки повинен мати імено index.html. Імена інших файлів повинні відображувати зміст відповідних сторінок.

4. Графічні файли зберігати в окремій папці з іменом *Images*.

5. Одна зі сторінок повинна містити резюме (з іменом *curriculum vitae*). Деякі основні пункти, які можуть бути зазначені в даному документі:

прізвище, ім'я, по батькові;

вік, стать, фото;

контакти (телефон домашній, телефон мобільний, факс, *e-mail*, веб-сторінка);

цілі/посади, мінімальний розмір заробітної плати (на які претендує подавець резюме);

освіта (рік закінчення, навчальний заклад, кваліфікація за дипломом – вказуються в зворотній хронологічній послідовності);

знання мов (мова, навички читання, письма, розмови);

професійні навички (операційні системи, середовища програмування, мови програмування, апаратні платформи);

сертифікати професійної діяльності;

досвід роботи (стаж роботи, установа, посада, коло обов'язків/участь у проектах – вказуються в зворотній хронологічній послідовності);

відомості про нагороди (перемоги в конкурсах, олімпіадах тощо);

хобі;

сімейний стан, кількість і вік дітей.

6. Використовуючи будь-який текстовий редактор або HTML-редактор, створити файли для сторінок розроблюваного сайту та зберегти їх на диску комп'ютера.

7. Перевірити сайт на валідність і виправити помилки.

8. У якості веб-технологій у процесі розробки сайта *використовувати тільки HTML!*

1.5. Контрольні запитання

1. З яких частин складається документ HTML?

2. У чому полягають правила сумісності синтаксису HTML з XML?

3. Чим відрізняються логічне та фізичне форматування тексту в документі HTML?

4. Як використовується службова інформація в блоці заголовка документа HTML?

5. Назвіть основні елементи структури сторінки.

6. Які типи списків можуть зустрічатися в документі HTML?

7. За яких умов для зображення краще вибрати один з графічних форматів GIF або JPG?

8. Чому слід завжди визначати альтернативний текст для зображень?

9. Що таке валідація документів?

Лабораторна робота № 2
Розроблення веб-сайта з використанням CSS та табличного верстання

2.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення основ каскадних таблиць стилів і табличного верстання веб-сайтів.

2.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити основні положення каскадних таблиць стилів і табличного верстання веб-сайтів.

2.3. Загальні тези лабораторної роботи

Верстання веб-сторінок називають створення такого HTML-коду, який дозволяє розміщувати елементи веб-сторінки (зображення, текст, лінії тощо) у потрібних місцях документа та відображувати їх у вікні браузера згідно з розробленим макетом. При цьому слід брати до уваги обмеження, властиві HTML і CSS, враховувати особливості браузерів і знати прийоми верстання, які дають бажаний результат [4; 6; 11].

Щоб стало зрозуміло, що ж входить у верстання і чому воно вважається досить важливим і складним етапом створення сайту, слід розібрати невеликий приклад. Необхідно відзначити, що верстання – це процес творчий і чітких алгоритмів тут не існує, тому не слід сприймати викладені поради та рекомендації як догму, це лише один з можливих шляхів.

Спочатку дизайнер готує макети веб-сторінок в графічному редакторі (наприклад, Adobe Illustrator, Adobe Photoshop), затверджує їх у замовника та передає верстальнику на формування HTML-коду.

Верстальник отримує роботу у вигляді набору малюнків, де кожнен з них відповідає макету окремої сторінки зі своїм дизайном. Наприклад, вигляд головної сторінки показаний на рис. 2.1.

Рис. 2.1. **Зображення головної сторінки після роботи дизайнера**

Тепер необхідно проаналізувати малюнок і вирішити, як його перетворити на веб-сторінку. Для зручності здійснюється логічне розбиття малюнка на окремі блоки, з якими йде подальша робота. Так, за рис. 2.1 можна виділити два великі блоки – "шапка" сторінки й основний контент. Спочатку слід розглянути "шапку", показану на рис. 2.2. За задумом дизайнера, кольорова та біла смуги вгорі повинні займати всю ширину веб-сторінки, а набір піктограм із заголовком сайта вирівнюється по центру вікна браузера.

Розташування малюнків відносно один одного мінятися не повинно, і кожен з них є посиланням на певний розділ сайта. З урахуванням зазначених особливостей можливі наступні варіанти.

Рис. 2.2. **Зображення "шапки" головної сторінки**

1. Зробити один малюнок і застосувати до нього карту-зображення.

2. Розрізати зображення на фрагменти й об'єднати їх воєдино за допомогою таблиці, при цьому окремі фрагменти будуть слугувати посиланням.

3. Скористатися позиціонуванням елементів.

Кожен з наведених методів хоча і призводить до потрібного результату, має також і свої недоліки, про які верстальник повинен мати уявлення.

Після того як перший блок буде готовий і втілений в HTML, можна переходити до роботи над наступним блоком. Тут тепер уже фігурує текст, тому відбувається формування стильового файлу, в якому діють наступні чинники:

колір фону веб-сторінки;

гарнітура основного шрифту, його розмір і колір;

розмір тексту окремих модулів (новин, наприклад);

колір, розмір і гарнітура шрифту заголовків;

параметри горизонтальних ліній і рамок.

Далі верстальник, використовуючи створений CSS-файл, створює остаточний HTML-документ головної сторінки.

На цьому починаються трудності, бо дизайнер підготував макет не тільки головної сторінки, а й макети решти розділів, які дещо відрізняються за своїм виглядом від уже виконаної роботи (рис. 2.3).

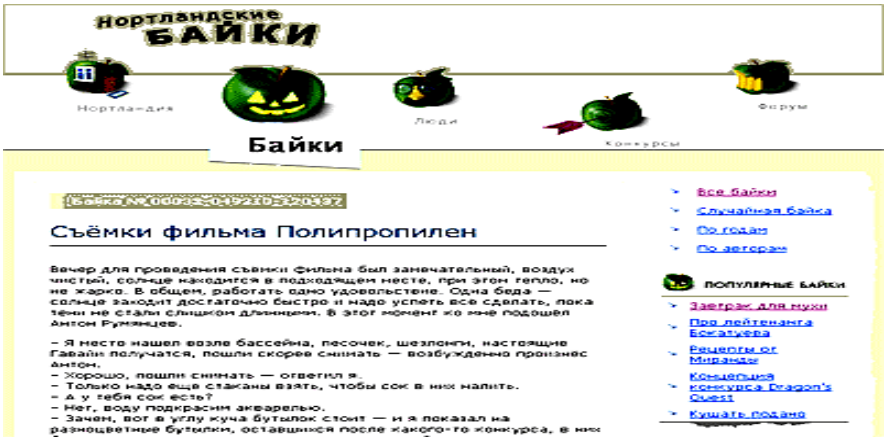


Рис. 2.3. **Зображення макета для розділу "Байки"**

"Шапку" тепер належить переробити, а в CSS включити параметри наново виниклих елементів. І так за кожним розділом.

Під час роботи над шаблонами і після її закінчення відбувається перевірка, яка повинна відповісти на такі питання:

чи коректно відображуються сторінки в популярних браузерах;

чи відбувається збереження цілісності даних після зміни розміру шрифту в браузері як у більший, так і менший бік;

чи можна продовжувати роботу з сайтом, якщо відключити показ зображень;

наскільки роздільна здатність монітора впливає на вигляд сторінок?

Слід також врахувати, що статті можуть мати різний обсяг і веб-сторінка повинна зберігати свій вигляд незалежно від нього.

Якщо помилки знайдені, то в шаблони вносяться виправлення з їх урахуванням, і так доки кількість помилок не буде зведена до мінімуму.

Зауваження

Верстальник, як правило, не створює всі веб-сторінки сайту, а робить кілька типових документів, які називаються шаблоном (темплейт, якщо використовувати жаргонний вислів). Після чого сторінки формуються за ним, динамічно за допомогою серверної програми, або заповненням шаблону текстом і збереженням у різних файлах.

Будь-який сайт – це комплексний продукт у тому сенсі, що він одночасно повинен вирішувати безліч часом суперечливих завдань. Так, сайт повинен бути естетичним і привабливим у плані дизайну, містити цікаву та корисну інформацію. Водночас він має легко індексуватися

пошуковими системами, швидко завантажуватися, а для автора – бути зручним для редагування та розширення можливостей. Сайт повинен також без явних помилок відображуватися в різних браузерах, підлаштовуватися під розмір вікна та "терпляче зносити" різні налаштування користувачів.

Отже, дизайн і матеріали сайту знаходяться поза сферою діяльності верстальника, але все інше так чи інакше підвладне саме йому. Хоча роботи верстальника не видно, оскільки вона прихована в коді документа, куди заглядають далеко не всі, саме вона забезпечує правильність відображення веб-сторінок і швидкість їх завантажування.

Друкарське мистецтво, або типографіка, – ось ще одна сфера, яка також відноситься до теми верстання. Це саме мистецтво, що об'єднало в собі риси дизайну та верстання, і ґрунтується воно на знанні шрифтів, досвіді з їх використання, психології сприйняття, почутті стилю тощо. Дійсно, який шрифт краще вибрати, щоб текст добре читався, легко сприймався при зміні розміру букв і робив сайт неповторним? При підготовці поліграфічних матеріалів відповідь на ці запитання покладають на дизайнера, але при створенні сайтів дизайнер і верстальник, як правило, вирішують його спільно.

Це пов'язано з початковою обмеженістю HTML у плані роботи зі шрифтами. Наприклад, використовувати можна тільки декілька універсальних гарнітур, які є на всіх комп'ютерних платформах. Зрозуміло, можна вказати будь-який шрифт, встановлений в операційній системі комп’ютера і він коректно буде показаний на даному комп'ютері. Але нема гарантії, що такий саме шрифт знаходиться на віддаленому комп'ютері користувача.

CSS розширив можливості HTML по роботі з текстом на веб-сторінці, але вони ще поступаються розвиненим програмам верстання поліграфічних матеріалів. Тому доводиться знати атрибути CSS, які відносяться до верстання тексту, особливості їх застосування, наявні обмеження та способи їх обходу. Але це вже завдання верстання.

Особливості верстання веб-сторінок

Хоча прабатьком верстання веб-сторінок є верстання поліграфічних матеріалів, між ними є одна важлива відмінність. Поліграфія буклету, листівки чи брошури друкується на аркушах встановленого розміру, у межах одного тиражу та має незначні або

навіть не помітні оком відмінності. Веб-сторінка же запускається на комп'ютері під управлінням клієнтської програми, званої браузером. Зрозуміло, що операційна система, її налаштування та власне сам браузер відрізняються від комп'ютера до комп'ютера. Із цього випливає банальний висновок, що один і той же документ сайту по-різному відображується у кожного користувача.

Відтак постає питання: чи можна зробити так, щоб веб-сторінка відображувалась однаково? Враховуючи, скільки доведеться взяти до уваги неоднозначних чинників, слід відповісти: ні, не можна. Тому завдання верстання веб-сторінок формулюється так: сформувати документ, який би коректно відображувався з невеликими відмінностями на основних платформах і в браузерах. Коректно це означає, що дотриманий вихідний задум автора показується в браузері без помилок.

Щоб реалізувати зазначене завдання, треба розуміти, як взагалі відбувається верстання веб-сторінок, і порівняти свої ідеї з їх виконанням. Далі описані деякі особливості, які мають велике значення при верстанні документів сайту [9 – 12].

Ширина документа

Спочатку розробнику сайта ширина вікна браузера користувача невідома, оскільки вона може змінюватися в найширших межах. Ширина залежить від дозвільної здатності монітора, довжини його діагоналі, розміру вікна і ще деяких варійованих даних. Отже, передбачити її заздалегідь простими засобами не можна. Із урахуванням цієї особливості узвичай є два способи верстання: фіксований і "гумовий".

Фіксований макет

У даному випадку слід діяти від зворотного та встановити загальну ширину макета, жорстко заданого та дорівненого до певної величини. Згідно зі загальносвітовою (табл. 2.1): www.w3schools.com та українською статистикою (рис. 2.4): www.ranking.com.ua, користувачі переважно використовують роздільну здатність монітора у 1024x768 пікселів або більше. Якщо брати за орієнтир 1 000 пікселів, загальна ширина макету за вирахуванням вертикальної смуги прокрутки та рамки браузера становитиме 800x990 пікселів. Тому слід орієнтуватися на цей розмір і встановлювати ширину макета, наприклад 900 пікселів.

Таблиця 2.1

Загальносвітова статистика роздільної здатності екранів користувачів

	Роздільна здатність екранів	%	Гістограма
1	1366x768 HD	22.84	
2	1920x1080 16:9 HD	10.78	
3	1024x768 4:3 XVGA	9.22	
4	1280x1024 5:4 SXGA	8.27	
5	1440x900 8:5 WSXGA	6.29	
6	1280x800 8:5 WXGA	5.96	
7	1600x900 16:9 HD+	5.17	
8	1680x1050 8:5 SXGA+	3.28	
9	768x1024	2.77	
10	1360x768	2.31	
11	1024x600 WSVGA	1.65	
12	1920x1200 8:5 WUXGA	1.22	
13	320x568	1.21	
14	1280x720 16:9 HD 720	1.17	
15	320x480 2:3 HVGA	1.07	
16	2560x1440	0.90	
17	Решта	17	

Перевага такої схеми в наступному. Оскільки загальна ширина макета точно відома, то можна легко підігнати під неї дизайн і створити зображення вже відомої ширини. У цілому подібне верстання наближається до верстання друкарського буклета: в обох випадках ширина носія інформації строго задана, за рахунок чого верстання хоча і частково, але спрощується (табл. 2. 2).

Таблиця 2.2

Тенденція змін роздільної здатності екранів користувачів за роками

Дата	Вище	1024x768	800x600	640x480	Невідоме
Січень 2014	94 **%**	5,5 **%**	0 %	0 %	0,5 %
Січень 2013	90 %	9 %	0,5 %	0 %	0,5 %
Січень 2012	85 %	13 %	1 %	0 %	1 %
Січень 2011	85,1 %	13,8 %	0,6 %	0 %	0,5 %
Січень 2010	76 %	20 %	1 %	0 %	3 %
Січень 2009	57 %	36 %	4 %	0 %	3 %
Січень 2008	38 %	48 %	8 %	0 %	6 %
Січень 2007	26 %	54 %	14 %	0 %	6 %
Січень 2006	17 %	57 %	20 %	0 %	6 %
Січень 2005	12 %	53 %	30 %	0 %	5 %
Січень 2004	10 %	47 %	37 %	1 %	5 %
Січень 2003	6 %	40 %	47 %	2 %	5 %
Січень 2002	6 %	34 %	52 %	3 %	5 %
Січень 2001	5 %	29 %	55 %	6 %	5 %
Січень 2000	4 %	25 %	56 %	11 %	4 %

Недолік, який приписують цьому виду верстання, фактично один – недостатньо ефективне використання вільної площі. Дійсно, для монітора з великою діагоналлю або високою роздільною здатністю екрану документ виглядатиме по-іншому, ніж на передбачуваних 800-х пікселях. На рис. 2.4 показано, як виглядає макет у такому випадку.

Рис. 2.4. **Макет фіксованої ширини, розміщений ліворуч**

Справа з'являється широка порожня смуга, розмір якої залежить від роздільної здатності екрану користувача та діагоналі його монітора. Щоб хоч якось зменшити порожній простір, макет зазвичай розташовують по центру вікна браузера.

"Гумовий" макет

Цей вид макета ґрунтується на тому, що в якості однієї з одиниць виміру виступають відсотки. Спільна робоча ширина вікна браузера – 100 %, і колонки макету в сумі не повинні її перевищувати, тому для зручності, як правило, застосовують відсотковий запис. У разі зміни розмірів вікна відбувається переформатування даних сторінки, щоб вони вписалися в нову ширину (рис. 2.5).

Рис. 2.5. **Веб-сторінка займає всю відведену ширину**

Цей вид верстання набуває все більшої популярності, і практично всі відомі сайти обрали саме його в силу того, що ефективно задіюється вся площа веб-сторінки. Але слід зазначити деякі особливості та недоліки, властиві "гумовому" верстанню.

Хоча веб-сторінка і підлаштовується під ширину вікна браузера, з досягненням деякої величини читати текст стає незручно – рядки дуже довгі й очі втомлюються при зчитуванні. Але, браузер можна згорнути у вікно, підібравши комфортний розмір до нього.

Верстати "гумовий" макет складніше, ніж аналогічний, але фіксованої ширини. Це пов'язано з тим, що доводиться враховувати безліч додаткових чинників і знати деякі прийоми верстання. До того ж популярні браузери неоднозначно трактують деякі параметри і в них

"гумовий" макет може відображуватися по-різному. Отже, знати про правила верстання треба якнайбільше.

Будь-який макет має певну мінімальну ширину, з досягненням якої веб-сторінка "розсипається" або з'являється горизонтальна смуга прокрутки. Наприклад, якщо в документ вставлений малюнок шириною 600 пікселів, то при зменшенні вікна до цієї величини браузер почне відображувати смугу прокрутки.

"Гумовий" дизайн характеризується активним використанням фонових зображень, які по горизонталі збираються без швів встик. Дійсно, змінити без втрати якості ширину малюнків не можна, а ось "підкласти" під них фон можна завжди.

Висота документа

Історично склалося, що гортання великого документа на комп'ютері відбувається зверху вниз. Для зручності перегортання призначені вертикальні смуги прокрутки, клавіатурні комбінації, колесо прокрутки миші. Переміщення ж по горизонталі відбувається не так зручно, тому горизонтальної смуги прокрутки бути не повинно. З цього випливає, що веб-сторінка повинна вписуватися у вікно браузера по ширині, але не по висоті, яка може змінюватися в дуже широкому діапазоні. На рис. 2.6 видно, що реальна висота документа не обмежена рамками браузера, але у вікні показана лише частина сторінки.

Рис. 2.6. **Відображення документа у вікні браузера по висоті**

У принципі, чим більше на сторінці інформації і, отже, більша висота документа, тим складніше знаходити потрібні дані. Тому текст структурують, розбивають на блоки та кожному з них дають свій заголовок, щоб погляду читача було за що зачепитися.

Також слід врахувати, що обсяг статей на сайті може досить сильно різнитися між собою. При цьому мінятиметься і висота сторінки, тому слід заздалегідь потурбуватися про те, щоб макет відображувався без помилок, незважаючи на різні значення висоти.

Об'єкти веб-сторінки прямокутні

Усі об'єкти на веб-сторінці мають прямокутну форму. Цей простий постулат не завжди узгоджується з тим, що можна побачити в дійсності, тому слід зробити пояснення. На відміну від традиційного верстання (йдеться про поліграфічні матеріали), де в документ можна вставляти будь-які об'єкти, в тому числі і векторні фігури довільної форми, верстання веб-сторінок має ряд обмежень. До обмежень відноситься і те, що додаються об'єкти прямокутні. Причому це стосується до їх форми, а не до вмісту, завдяки чому необхідний дизайн можна конструювати за допомогою набору зображень. На рис. 2.7 показано малюнок дещо неправильної форми. Але через те, що фон у цього малюнка збігається з фоном веб-сторінки, вихідна прямокутність зображення не видна.

Рис. 2.7. **Зображення на малюнку може мати довільну форму**

Проте варто додати навколо малюнка рамку, яка допомагає зрозуміти, що зображення прямокутне (рис. 2.8). Наприклад, якщо включити обтікання малюнка текстом, то він буде обходити малюнок саме по межі рамки.

Рис. 2.8. **Але сам малюнок залишається прямокутним**

Дана особливість породила техніки, пов'язані з верстанням, про які йдеться далі.

Активне використання малюнків

Малюнки застосовуються не тільки для ілюстрації тексту, а й виконують на сайті багато різноманітних ролей, наприклад, використовуються для створення привабливого дизайну, слугують розпіркою між осередками таблиці, створюють градієнтні заливки, фонові зображення тощо.

Розрізання зображення на фрагменти

Малюнок може займати дуже велику вихідну площу. Слід розрізати його на прямокутники, щоб отримати чудовий конструктор, в якому одні фрагменти зображення допускається замінювати текстом, інші – анімацією, а треті модифікувати "на льоту". Таким чином, це буде засіб для обходу прямокутної природи зображень, адже в "склеєному" малюнку може бракувати деяких фрагментів, наприклад куточків.

Застосування фонового малюнка

Фоновий малюнок зручний тим, що він може заповнювати всю відведену ширину або висоту під блоком. Це дозволяє створювати лінії або інші декоративні елементи, які прив'язуються до ширини або висоти тексту і не залежать від розмірів вікна. До того ж поверх фону можна накладати текст, що також розширює можливості дизайну веб-сторінок.

Зображення замість тексту

Якщо засоби щодо верстання мають певні обмеження, то чому б не створити текст в графічному редакторі і не вставити його в якості малюнка або Flash? Це гарантує, що текст збереже свій вигляд і накреслення, попри зовнішні умови. Однак тут є і зворотний бік – малюнки займають більший обсяг, ніж у рядовому тексті, їх складніше правити, вони не індексуються пошуковими машинами, їх показ користувачі можуть відключити. Зображення на сайті хоча і застосовують замість тексту, але через значні недоліки досить обмежено, наприклад для заголовка сайту.

Модульні сітки

Модульна сітка – це набір невидимих напрямних, уздовж яких розташовуються елементи веб-сторінки. Це полегшує розміщення даних у документі, забезпечує візуальний зв'язок між окремими блоками та зберігає спадкоємність дизайну під час переходу від однієї сторінки до іншої. Веб-сторінка фактично розглядається як набір прямокутних блоків, які викладаються за певним порядком. При цьому, як правило, дані розташовуються по колонках, тому при верстанні застосовують термін одно-, дво-, триколонний макет тощо. Для прикладу слід розглянути головну сторінку сайту deviantart.com (рис. 2.9).

Кожен блок цієї сторінки чітко відокремлений від інших за допомогою порожнього простору, рамки або роздільника, в якості якого виступає кольоровий прямокутник із текстом заголовка. Але чи дійсно йдеться саме про три колонки? У деяких випадках одразу визначити, скільки колонок містить макет, справді важко. Слід розуміти, що колонки можуть об'єднуватися, а також містити не тільки суцільний текст, а й графічні вставки. Якщо подати основні блоки сторінки у вигляді однотонних прямокутників, то створюється наочна модульна сітка, за якою одразу стає зрозуміло, як зверстаний документ (рис. 2.10).

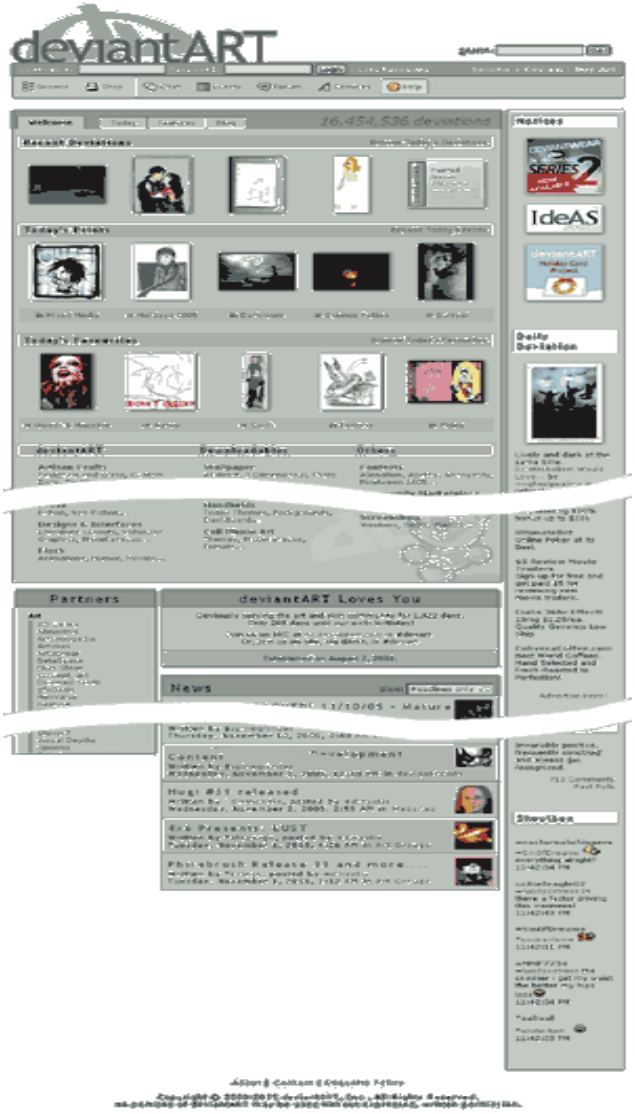


Рис. 2.9. **Головна сторінка deviantart.com**

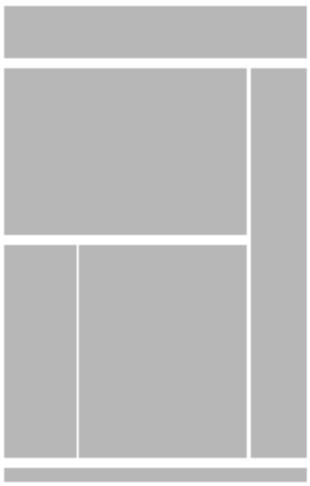


Рис. 2.10. **Модульна сітка для головної сторінки deviantart.com**

За даними малюнка видно, що верхній блок з назвою сайта, формою пошуку та кнопками навігації займає всю ширину сторінки. Далі йдуть три колонки, причому перші дві попередньо об'єднані в одну для зручності подання необхідної інформації. Завершує макет блок з контактною та правовою інформацією.

Принципи побудови модульної сітки такі. Спочатку макет веб-сторінки розробляють на аркуші паперу. Це дозволяє, не витрачаючи часу, швидко зробити серію нарисів і вже з них вибрати відповідний ескіз. Постає важливе питання: скільки варіантів можна створити за десять хвилин у графічному редакторі та скільки за цей же час за допомогою олівця та паперу? При цьому не має значення ступінь володіння програмою, адже на папері вийде швидше. Хоча ескіз може бути і коструватим, головне – щоб автор сам зрозумів, що він намалював. Зазвичай замість тексту та малюнків застосовують схематичні позначки. Наприклад, текст позначається декількома горизонтальними лініями (рис. 2.11), а малюнки зображуються затемненими блоками або перекресленими прямокутниками (рис. 2.12).

Рис. 2.11. **Позначення тексту в макетах**

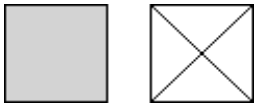


Рис. 2.12. **Позначення зображень в макетах**

Надалі ці позначення будуть застосовуватись для розгляду найбільш поширених модульних сіток.

Одноколонна сітка

Текст в одну колонку найчастіше зустрічається в академічному дизайні, за фіксованим макетом і в публікації великого тексту.

Зауваження

Академічний дизайн характеризується мінімалізмом оформлення та навіть аскетизмом. Основний упор робиться на змістовну частину, а дизайну як такому практично не приділяється уваги. Переважно академічний дизайн зустрічається в науковому середовищі.

На рис. 2.13 показана типова схема одноколонної модульної сітки. Як правило, спостерігаються чотири основні блоки – заголовок сторінки, набір посилань на інші сторінки сайту (навігація), власне сам текст і в самому низу – контактна інформація. Якщо висота сторінки досить велика, то блок навігації дублюють внизу або роблять посилання "Вгору", яке переміщує на початок документа.

Рис. 2.13. **Одноколонна модульна сітка**

Ілюстрації в тексті зустрічаються по ходу, текст зазвичай обтікає їх по контуру. Під час активного застосування зображень на сайті зручніше скористатися фіксованим макетом, ширина якого точно відома. Тоді можна готувати заданого розміру малюнки, які точно впишуться в макет сторінки. Наприклад, головна сторінка сайта hneu.edu.ua (рис. 2.14) заснована на макеті фіксованої ширини та практично повністю складається з набору малюнків.

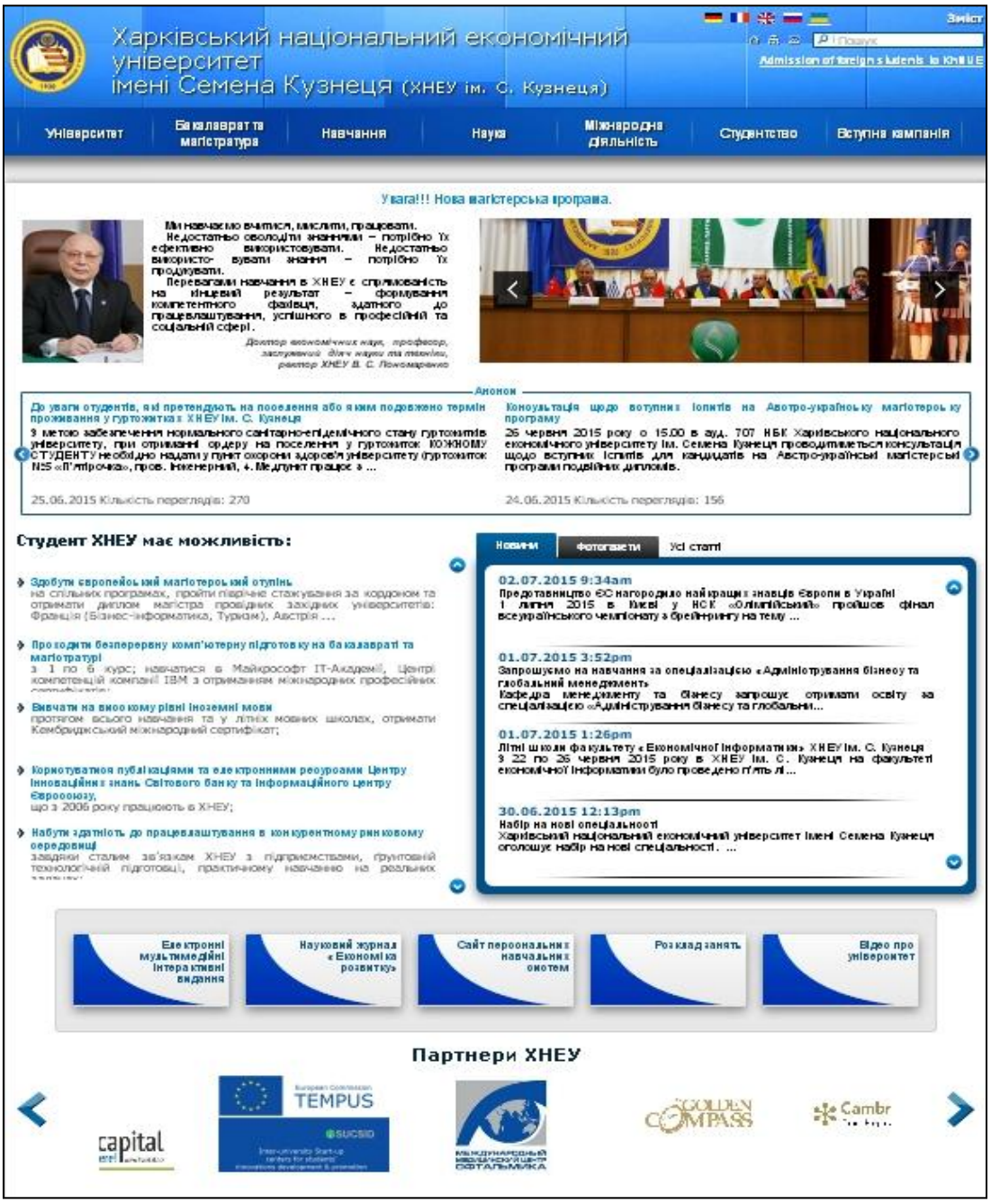


Рис. 2.14. **Головна сторінка сайта hneu.edu.ua**

Модульна сітка для цього сайта подана на рис. 2.15. Спочатку йде заголовок сайта з формою пошуку, нижче слідує навігація, а окремі фотографії є саморекламою з посиланнями на відповідні розділи сайта. У самому низу розташовані посилання на інформацію про сайт.

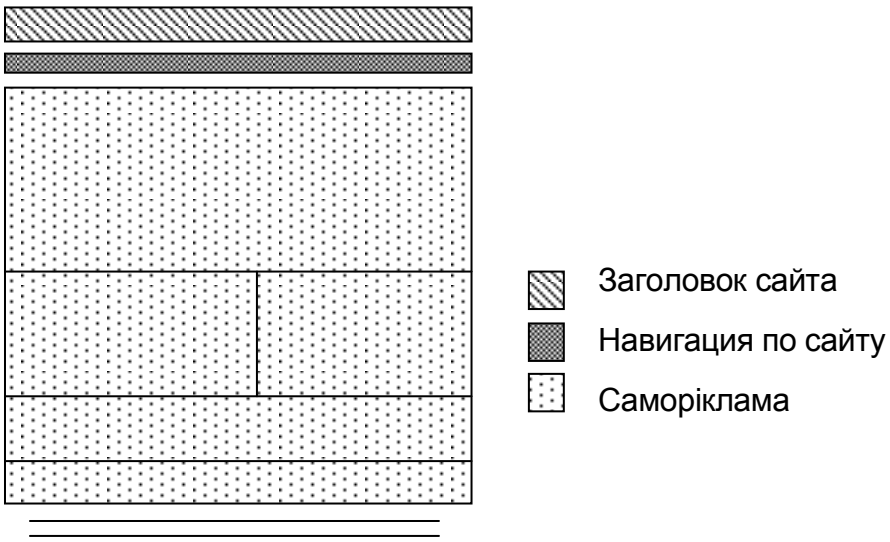


Рис. 2.15. **Модульна сітка для головної сторінки сайта hneu.edu.ua**

Двоколонна сітка

Це один із найпоширеніших варіантів, використовуванних на сайтах. У такій модульній сітці використовуються дві колонки – одна відводиться під основний текст, а друга використовується для навігації й іншої корисної інформації (рис. 2.16)

Рис. 2.16. **Двоколонна модульна сітка**

Принципового значення немає, ліворуч чи праворуч розташовується колонка з навігацією, зустрічаються і обидва варіанти.

У певному сенсі двоколонна сітка стала стандартом де-факто для інформаційних сайтів через свою зручність. Дійсно, все "під рукою" – і текст, і посилання, до того ж дана сітка не виключає підключення горизонтальної навігації, як це прийнято в одноколонній сітці.

Двоколонні сітки зручні при створенні найрізноманітніших сайтів і не вимагають особливих знань з верстання веб-сторінок. Єдиний недолік, який їм закидають, що подібні сайти виглядають дещо одноманітно. Але, з іншого боку, користувачам зручніше працювати з сайтом звичайного виду, без зайвих "наворотів".

Триколонна сітка

Такі сітки часто застосовуються на головних сторінках сайтів, де одночасно потрібно показати користувачеві безліч можливостей, які він виявить на даному сайті. Також триколонна сітка використовується і на внутрішніх сторінках, якщо для розміщення різноманітної інформації двох колонок не вистачає (рис. 2.17).

Рис. 2.17. **Триколонна модульна сітка**

Одна з колонок віддається під навігацію, друга, найширша, – під основний текст, а до третьої колонки додають рекламу, посилання, текст тощо.

Триколонна сітка забезпечує більше простору для дизайну, адже в деяких місцях можна об'єднувати колонки, розбивати матеріал на окремі фрагменти та візуально відокремлювати один блок від іншого. Макет може вийти досить складним, але результат того вартий. Стосовно до головної сторінки сайта deviantart.com розглядається більш детальна модульна сітка (рис. 2.18). Окремі блоки виділені різними кольорами.

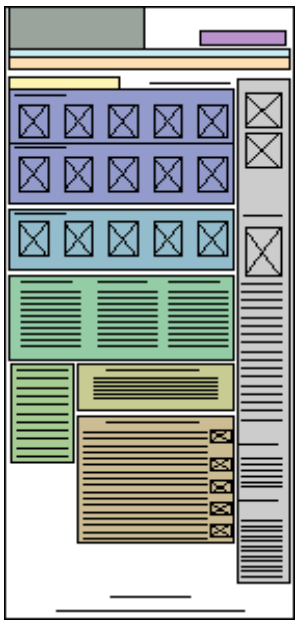


Рис. 2.18. **Модульна сітка для головної сторінки deviantart.com**

На сайті deviantart.com застосовуються три колонки, дві з них часто об'єднуються для отримання більш широкої області. Це виправдано, оскільки потрібно розмістити п’ять фотографій або три колонки з текстом.

До недоліків триколонної сітки відносять достатню складність верстання макета. Щоб отримати потрібний результат, доводиться затрачувати багато часу на створення стильового файла та налагодження документа в різних браузерах.

Існують також інші види модульних сіток, наприклад багатоколонні. Однак вони практикуються досить рідко через обмежену ширину вікна браузера. У такому випадку текст доводиться робити дрібним або застосовувати інші способи, щоб не виникло горизонтальної смуги прокрутки. Від цього може постраждати зручність сприйняття інформації користувачем.

Модульна сітка не є єдиним варіантом при верстанні веб-сторінок. Можна скористатися перспективою, хаотичністю або іншою структурою, головне – щоб вона була естетичною та привабливою. Відмова від правил верстання характерна для дизайн-студій, завдання яких полягає в тому, щоб вразити своїм сайтом уяву відвідувачів і привернути до себе їх увагу. Але навіть вони роблять для замовника сайти на основі формальних сіток, оскільки так простіше додавати інформацію та вносити різноманітність в дизайн макета.

Основи каскадних таблиць стилів (CSS)

Звичайний HTML дозволяє задавати колір і розмір тексту за допомогою тегів форматування. Якщо знадобиться змінити параметри однотипних елементів на сайті, доведеться переглядати всі сторінки, щоб знайти та поміняти теги.

Каскадні таблиці стилів (Cascading Style Sheets, CSS) дозволяють зберігати колір, розміри тексту й інші параметри в стилях. Стилем називається набір правил форматування, що застосовується до елемента документа, щоб швидко змінити його зовнішній вигляд.

Стилі дозволяють однією дією застосувати одразу всю групу атрибутів форматування. За їх допомогою можна, наприклад, змінити вигляд усіх заголовків. Замість форматування заголовка в три прийоми, коли спочатку задається його розмір, потім – шрифт "Arial" і, нарешті, вирівнювання по центру, те ж саме можна зробити одночасно, застосувавши стиль до тегу <H1>. Якщо потрібно швидко змінити зовнішній вигляд тексту, створеного за допомогою одного зі стилів, достатньо змінити параметри стилю у всіх документах, де він використовується, і вигляд тексту зміниться автоматично.

Інша перевага CSS полягає в тому, що стилі пропонують набагато більше можливостей для форматування, ніж простий HTML. Крім того, стилі можуть зберігатися в зовнішньому файлі, браузер кешує такі документи, тому завантаження сайту буде відбуватися дещо швидше.

CSS є потужною системою для розробників сайтів, розширюючи їх можливості з дизайну та верстання веб-сторінок. У науковому середовищі, звідки пішла родом технологія WWW, фахівці були зайняті переважно змістом документів, ніж їх оформленням. Однак для більшості з них уявлення про сайт, про те, як він виглядає, відіграє більш важливу роль. Обмеження HTML породили безліч технік створення веб-сторінок – таких, як:

використання різноманітних розширень HTML;

застосування зображень замість тексту;

використання малюнків для контролю порожнього простору, так звані "розпірки";

використання таблиць для верстання веб-сторінок;

написання програмних скриптів замість використання HTML.

Ці техніки ускладнюють розроблення веб-сторінок, пропонують обмежену гнучкість в їх створенні й управлінні, а також створюють трудності для людей, які ними не володіють.

Стилі вирішують ці проблеми, замінюючи водночас лише обмежену галузь механізмів подання HTML.

Таблиці стилів можуть бути додані на веб-сторінку трьома різними способами, які розрізняються за своїми можливостями.

Таблиці пов'язаних стилів

Найпотужніший і зручний спосіб визначення стилів і правил для сайту. Стилі зберігаються в окремому файлі, який може бути використаний для будь-яких веб-сторінок.

Для підключення таблиці пов'язаних стилів використовується тег <LINK> у заголовку сторінки (рис. 2.19).

```
<!DOCTYPE HTML>
        <html>
        <head>
        <meta charset="utf-8">
        <title>Стилі</title>
        <link rel="stylesheet" type="text/css"
href="mysite.css">
        <link rel="stylesheet" type="text/css"
href="http://www.mysite.ru/main.css">
        </head>
        <body>
        <h1>Hello, world!</h1>
        </body>
</html>
```

Рис. 2.19. **Підключення таблиці пов'язаних стилів**

Шлях до файла зі стилями може бути як відносним, так і абсолютним, як показано на даному прикладі.

Переваги даного способу:

використовується один файл зі стилем для будь-якої кількості веб-сторінок, також можна його застосовувати на інших сайтах;

можна змінювати таблицю стилів без модифікації веб-сторінок;

під час зміни стилю в одному файлі стиль автоматично застосовується до всіх сторінок, де є на нього посилання. Це, безсумнівно, зручно. Якщо вказується розмір шрифту тільки в одному місці, він змінюється на всіх ста або більше веб-сторінках сайта;

файл із стилем під час першого завантаження поміщається в кеш на локальному комп'ютері окремо від веб-сторінок, тому завантаження сайта відбувається швидше.

Таблиці глобальних стилів

Стиль визначається в самому документі і зазвичай розташовується в заголовку веб-сторінки. За своєю гнучкістю та можливостям цей спосіб використання стилю поступається попередньому, але також дозволяє розміщувати всі стилі в одному місці; у даному випадку – прямо в тілі документа. Визначення стилю задається тегом <STYLE> (рис. 2.20).

```
<!DOCTYPE HTML>
        <html>
        <head>
        <meta charset="utf-8">
        <title>Стилі</title>
        <style type="text/css">
        H1 {
         font-size: 120%; /* Розмір шрифту */
         font-family: Verdana, Arial, Helvetica, sans-serif; /* Сімейство шрифту */
         color: #336; /* Колір тексту */
        }
        </style>
        </head>

        <body>
        <H1>Hello, world!</H1>
        </body>
</html>
```

Рис. 2.20. **Використання таблиці глобальних стилів**

У даному прикладі показана зміна стилю заголовка <H1>. На веб-сторінці тепер достатньо вказати тільки цей тег, і стилі будуть додані до нього автоматично.

Внутрішні стилі

Внутрішній стиль є, по суті, розширенням для одинарного тегу та використовується на веб-сторінці. Для визначення стилю використовується параметр style, а його атрибути вказуються за допомогою мови таблиці стилів (рис. 2.21).

Рекомендується використовувати внутрішній стиль для одинарних тегів або відмовитися від його використання взагалі, оскільки зміна ряду елементів стає проблематичною. Внутрішні стилі не відповідають ідеології структурного документа, коли вміст і його оформлення розділені.

```
<!DOCTYPE HTML>
        <html>

        <head>
        <meta charset="utf-8">
        <title>Стилі</title>
         </head>

         <body>
        <H1 style="font-size: 120%; font-family: Verdana, Arial, Helvetica, sans-
serif; color: #336">
        Hello, world!
        </H1>

        </body>
</html>
```

Рис. 2.21. **Використання внутрішніх стилів**

Усі описані методи використання CSS можуть застосовуватися як самостійно, так і сукупно. Необхідно пам'ятати про їх ієрархію. Першим завжди застосовується внутрішній стиль, далі – таблиця глобальних стилів і в останню чергу – таблиця пов'язаних стилів.

Текст в CSS

За допомогою CSS можна визначати стиль і вид тексту, аналогічно тому як використовується тег <FONT>, що задає властивості шрифту.

Але стилі володіють значними можливостями і дозволяють скоротити код HTML.

Властивості шрифту

Зміна вигляду шрифту і його розміру відбувається через властивості CSS, які описані в табл. 2.3.

Таблиця 2.3

Атрибути CSS для управління шрифтами

Властивості	Значення	Опис	Приклад
font-family	ім'я шрифту	Задає список шрифтів	P {font-family: Arial, serif}
font-style	normal italic oblique	Нормальний шрифт; курсив; похилий шрифт	P {font-style: italic}
font-variant	normal small-caps	Капітель (особливі малі літери)	P {font-variant: small-caps}
font-weight	normal lighter bold bolder 100-900	Нормальна жирність; світле накреслення; напівжирний; 100-світлий шрифт; 900-більш жирний	P {font-weight: bold}
font-size	normal pt px %	Нормальний розмір; пункти; пікселі; відсотки	font-size: normal font-size: 12pt font-size: 12px font-size: 120%

Властивості тексту

Крім зміни параметрів шрифтів, можна управляти і властивостями всього тексту. Значення властивостей наведено в табл. 2.4.

Таблиця 2.4

Властивості CSS для управління видом тексту

Властивості	Значення	Опис	Приклад
line-height	normal множник точно %	Інтерліньяж (міжрядковий інтервал)	line-height: normal line-height: 1.5 line-height: 12px line-height: 120%
text-decoration	none underline overline line-through blink	Прибрати всі оформлення; підкреслення; лінія над текстом; перекреслення; миготіння тексту	text-decoration: none
text-transform	none capitalize uppercase lowercase	Прибрати всі ефекти; починати з прописної; УСІ ПРОПИСНІ усі малі	text-transform: capitalize
text-align	left right center justify	Вирівнювання тексту	text-align: justify вирівнювання по ширині
text-indent	точно %	Відступ першого рядка	text-indent:15px; text-indent:10%

CSS має кілька опцій для визначення кольору тексту та фонових областей на веб-сторінці. Ці опції для роботи з кольором не тільки замінюють аналогічні в простому HTML, а й дають масу нових можливостей. Наприклад, традиційний шлях для створення кольорової області полягає в застосуванні таблиці. Стилі дозволяють відмовитися від подібного використання таблиць, пропонуючи більш прості та зручні варіанти управління кольором.

У табл. 2.5 показані властивості елементів, призначених для задання кольору.

Таблиця 2.5

Управління кольором фону та тексту

Властивості	Значення	Опис	Приклад
color	Колір	Встановлює колір тексту	P { color: #330000 }
background-color	Колір transparent	Колір фону	BODY { background-color: #6699FF }
background-image	URL none	Фоновий малюнок	BODY { background-image: url (bg.gif) }
background-repeat	repeat repeat-x repeat-y no-repeat	Повторюваність фонового малюнка	BODY { background-image: url (bg.gif) background-repeat: repeat-y }
background-attachment	scroll fixed	Прокручуваність фону разом із документом	BODY { background-image: url (bg.gif) background-attachment: fixed }
background-position	Відсотки Пікселі top center bottom left right	Початкове розташування фонового малюнка	BODY { background-position: left top }

2.4. Завдання до лабораторної роботи

1. Зверстати шаблон сторінок сайта, використовуючи прийоми табличного верстання, згідно з варіантом завдання.

2. Під час створення шаблону строго витримати принцип поділу структури (застосовувати HTML) і подання (застосовувати тільки CSS).

3. Тип макета (каркаса) шаблону повинен бути з фіксованою шириною. Ширину макету вибрати на підставі табл. 2.1 і 2.2.

4. Перевірити шаблон на відповідність до стандартів W3C і на сумісність з найбільш популярними в Україні та світі браузерами.

5. На підставі шаблону створити сайт, що складається мінімум з п’яти сторінок.

6. Тематика сайту – довільна.

7. Вимоги до структури каталогу сайта мають відповідати до вимог лабораторної роботи № 1.

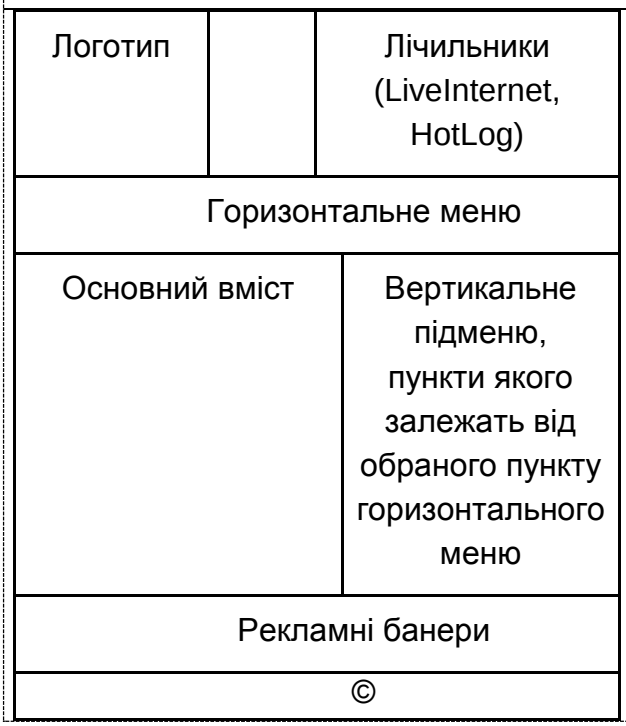
8. На захисті лабораторної роботи надати сайт і шаблон.

Варіанти завдання

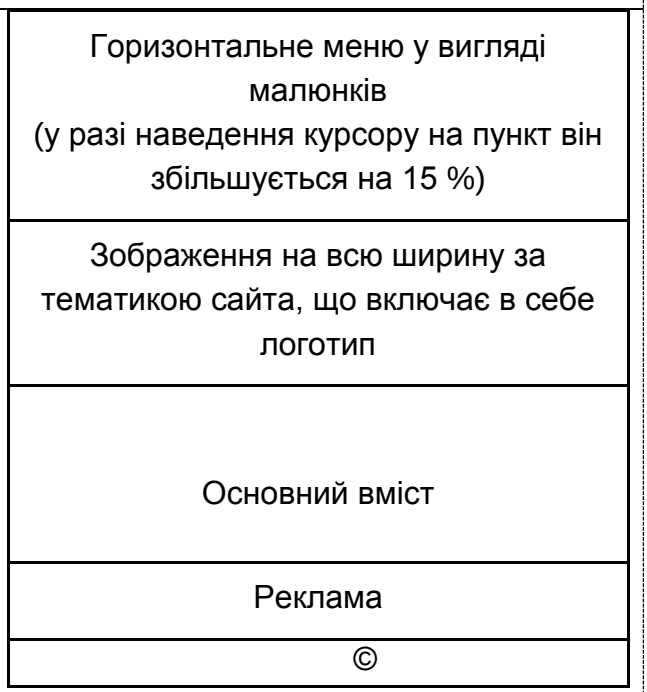
Варіант 1

Варіант 2

Варіант 3



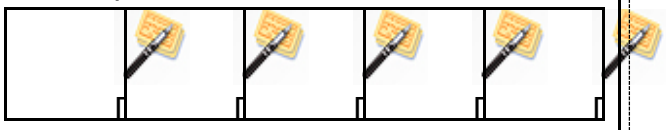
Варіант 4



Варіант 5

<table>
<tr><td>Логотип</td><td colspan="2">Пошук</td></tr>
<tr><td rowspan="3">Вертикальне головне меню</td><td>Малюнок поточного пункту (сторінки)</td><td>Контекстна реклама</td></tr>
<tr><td>Повна назва поточної сторінки</td><td>Новини</td></tr>
<tr><td>Основний вміст</td><td>новина 1
–
новина 2
–
новина 3</td></tr>
<tr><td>Контактна інформація</td><td colspan="2">Партнери (в логотипах партнерів)</td></tr>
<tr><td colspan="3">Горизонтальне головне меню</td></tr>
<tr><td colspan="3">© 2010</td></tr>
</table>

Варіант 6

<table>
<tr><td colspan="2">Назва компанії. Місія компанії</td></tr>
<tr><td colspan="2">Горизонтальне головне меню:
  </td></tr>
<tr><td>Зображення за темою поточної сторінки</td><td rowspan="2">Основний вміст</td></tr>
<tr><td>Останні новини
–
–
–</td></tr>
<tr><td colspan="2">Пункт 1 | Пункт 2 | Пункт 3 | Пункт 4 | Пункт 5</td></tr>
<tr><td colspan="2">©</td></tr>
</table>

Варіант 7

<table>
<tr><td>Логотип</td><td>Головне вертикальне меню</td><td rowspan="2">Малюнок за темою сайта</td></tr>
<tr><td colspan="2">Розширена назва поточної сторінки. Анотація (резюме) поточної сторінки</td></tr>
<tr><td rowspan="2">Вертикальне підменю</td><td colspan="2">Новини & Події</td></tr>
<tr><td>новина 1
–
новина 2
–
новина 3</td><td>подія 1
–
подія 2
–
подія 3</td></tr>
<tr><td rowspan="2">Детальні контактні реквізити</td><td colspan="2">Основний вміст</td></tr>
<tr><td colspan="2">Горизонтальне головне меню</td></tr>
</table>

Варіант 8

<table>
<tr><td>Логотип</td><td>Основні сторінки у вигляді піктограм</td></tr>
<tr><td>Вертикальне меню</td><td>Зображення за темою сайта</td></tr>
<tr><td>Розгорнута назва поточної сторінки</td><td>Анотація (резюме) поточної сторінки</td></tr>
<tr><td rowspan="3">Події
–
–
–</td><td>Основний вміст</td></tr>
<tr><td>Горизонтальне головне меню</td></tr>
<tr><td>©</td></tr>
</table>

Варіант 9

<table>
<tr><td>Контактні реквізити</td><td>Слоган (девіз компанії)</td><td>Логотип</td></tr>
<tr><td>Пошук</td><td colspan="2">Горизонтальне меню</td></tr>
<tr><td>Малюнок за темою сайта</td><td colspan="2" rowspan="2">Вертикальне підменю</td></tr>
<tr><td>Основний вміст</td></tr>
<tr><td>Горизонтальне головне меню</td><td colspan="2" rowspan="2">Новини
–
–
–

Статті
–
–
–</td></tr>
<tr><td>© 2010</td></tr>
</table>

Варіант 10

<table>
<tr><td rowspan="2">Зображення з тематики сайта, що включає в себе логотип</td><td colspan="2">Горизонтальне головне меню</td></tr>
<tr><td>Розгорнута назва поточної сторінки</td><td>Зображення за темою поточної сторінки</td></tr>
<tr><td colspan="2" rowspan="2">Основний вміст</td><td>Новини</td></tr>
<tr><td>Новина 1
–
Новина 2
–
Новина 3</td></tr>
<tr><td>Контактна інформація</td><td colspan="2">Пошук на сайті</td></tr>
<tr><td colspan="3">Горизонтальне головне меню</td></tr>
<tr><td colspan="3">© 2010</td></tr>
</table>

2.5. Контрольні запитання

1. Що таке верстання web-сайта?
2. Які типи верстання ви знаєте?
3. Що таке модульна сітка?
4. Назвіть переваги та недоліки табличного верстання.
5. Що таке CSS?
6. Назвіть способи, якими можуть бути додані таблиці стилів до веб-сторінки.

Лабораторна робота № 3
Розроблення веб-сайтів із використанням блокового верстання

3.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення основ каскадних таблиць стилів і блокового верстання (шарами) веб-сайтів.

3.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити основні положення каскадних таблиць стилів і табличного верстання веб-сайтів.

3.3. Загальні тези лабораторної роботи

Шари ввела компанія Netscape, включивши в свій браузер підтримку тегу <LAYER>. Цей тег дозволяв приховувати/показувати поточний вміст, встановлювати розташування щодо вікна браузера, накладати один шар поверх інших і завантажувати дані у вміст шару з файлу. Що характерно, всі ці параметри легко мінялися за допомогою JavaScript, і це розширювало можливості зі створення дійсно динамічного контенту на сторінці. Незважаючи на потужний набір можливостей, тег <LAYER> не був включений в специфікацію HTML і залишився лише розширенням браузера Netscape [7; 9; 10; 12].

Однак необхідність у зазначених можливостях, які б застосовувалися на сайтах, вже назріла, і наприкінці 1996 року синтаксис для роботи з шарами був розроблений і схвалений у робочому проекті консорціуму "CSS Positioning (CSS-P)". Основне навантаження лягало на стилі, з їх допомогою можна управляти виглядом будь-якого елементу, в тому числі динамічно змінювати значення через JavaScript. На жаль, об'єктні моделі браузерів щодо доступу до елементів різнилися, тому доводилося писати достатньо складний код, який би враховував ці відмінності.

На сьогодні розробники популярних браузерів дотримуються специфікацій HTML і CSS, що сильно полегшило роботу творцям сайтів, оскільки скоротило час на налагодження сайту в різних браузерах. Однак відмінності в підходах до браузерів існують, і при їх виникненні розробники дотримуються наступних форм роботи.

Якщо спостерігаються невеликі відмінності у відображенні одного сайта в різних браузерах, то ці відмінності ігнорують, тобто не виправляють. Тут варто зробити застереження, що сайт у будь-якому випадку повинен відображуватися коректно та без помилок.

Якщо у сайта є істотні відмінності під час його показу в тому чи іншому браузері, то для їх усунення застосовують хаки.

Хак – це набір прийомів, коли певному браузеру "підсовують" код, який розуміється тільки цим браузером, а іншими ігнорується.

Незважаючи на те, що хаки працюють, використовувати їх слід обмежено або взагалі обходитися без них. Справа в тому, що хаки знижують універсальність коду, і для модифікації параметрів одного елемента доводиться вносити зміни одночасно в різних місцях.

Є й інший, перспективний шлях – дотримуватися специфікації CSS. Незважаючи на те, що браузери не повною мірою її підтримують, вони прогресують саме в цьому напрямі, а саме – до повної підтримки різних специфікацій (HTML, CSS, DOM). Таким чином, майбутні версії браузерів будуть уніфіковані, і той самий сайт буде відображуватися коректно.

Доцільно повернутися до шарів. Зрозуміло, що вони безпосередньо пов'язані зі стилями. То, чи не є кожен з елементів HTML-коду, до якого додаються стилі, шаром? У певному сенсі так і є. Однак це внесло б неабияку плутанину, якби замість "таблиця" або "параграф" ми говорили "шар". Тому слід домовитись щодо застосування цього терміну тільки до тегів <DIV> і <SPAN>.

Шар – це елемент веб-сторінки, створений за допомогою тегів <DIV> або <SPAN>, до якого застосовується стильове оформлення.

Таким чином, верстання за допомогою шарів полягає в конструктивному використанні тегів <DIV> і стилів. При цьому дотримуються наступних принципів [5, 7].

Розділення вмісту й оформлення

Код HTML повинен містити тільки теги розмітки та логічного форматування, а будь-яке оформлення виноситься за межі коду в стилі. Такий підхід дозволяє незалежно управляти виглядом елементів сторінки й її вмістом. Завдяки цьому над сайтом може працювати декілька фахівців, при цьому кожен виконує свою функцію незалежно від інших. Дизайнер, верстальник і програміст працюють над своїми завданнями автономно, скорочуючи час на розроблення сайта.

Активне застосування тега DIV

При використанні шарів істотне значення приділяється універсальному тегу <DIV>, який виконує безліч функцій. Фактично це основа, на яку "навішуються" стилі, перетворюючи її то в іграшку, то в тваринку. Не обов’язково застосовувати тільки цей тег, адже потрібно вставляти малюнки й оформлювати текст. Але при верстанні за допомогою шарів тег <DIV> є складовою верстання, його фундаментом.

Завдяки тегу HTML-код розпадається на ряд чітких наочних блоків, за рахунок чого верстання шарами називається також блоковим верстанням. Код стає більш компактним, ніж за табличного верстання, до того ж пошукові системи краще його індексують.

Таблиці застосовуються тільки для подання табличних даних

Під час верстання шарами зазвичай використовують таблиці, але тільки коли вони потрібні, наприклад, для наочного відображення чисел та інших табличних даних. Варіант, коли від таблиць пропонується відмовитися взагалі, є недоцільним і, більше того, шкідливим.

Використання стилів не є обов'язковою характеристикою верстання шарами, і для табличного верстання стилі можуть застосовуватися досить активно. Це стає стандартом де-факто, і без стилів тепер працювати не можна.

Шари не таблиці

Незважаючи на банальність подібного твердження, більшість розробників на практиці його постійно спростовують. Але якщо усвідомити, що це різні інструменти й елементи конструктора, все стане простіше. Дійсно, таблиці та методи верстання з їх допомогою краще застосовувати в одному випадку, а шари – в іншому. Слід чітко розділяти підходи та принципи верстання. Тоді не знадобиться заганяти творчу думку в прокрустове ложе антагоністичної технології. Треба просто піти іншим шляхом.

Висота шарів обмежена висотою контенту

У таблиці сусідні осередки взаємопов'язані, тому висота у них однакова, незалежно від обсягу інформації. Це добре видно, якщо залити фон осередків різним кольором. Шари ж у певному сенсі незалежні один від одного, тому висота шару визначається його вмістом.

Вигляд документа при цьому буде відрізнятися від його табличного "побратима" (рис. 3.1).

Рис. 3.1. **Сторінка, створена за допомогою шарів**

Висота різних колонок на даному малюнку розрізняється, оскільки формується за рахунок їх вмісту.

За потребою, висоту шару можна встановити через стилі, використовуючи для цього різні одиниці – відсотки, пікселі, дюйми тощо. Але якщо вміст шару перевищує його задану висоту, то браузери по-різному інтерпретують цей факт: одні розширюють висоту шару під новий контент, а інші браузери, залишаючи висоту вихідної, накладають контент поверх шару.

Блокове верстання

Уже згадувалося, що шари в більшості випадків є незалежними один від одного, за рахунок чого вони, як окремі блоки, можуть додаватися або видалятися в макеті веб-сторінки. За таку поведінку верстання за допомогою шарів отримало назву "блокове верстання". Шари допустимо вкладати один в інший для формування бажаного декоративного елемента. Тому під терміном "блок" розуміють не стільки окремий шар, скільки їх сукупність.

Розташування колонок

За умовчанням, вміст контейнерів <DIV> на веб-сторінці розташовується по вертикалі: спочатку йде один шар, нижче розташовується

наступний тощо. Під час створення колонок потрібно розташовувати шари поряд по горизонталі, для чого застосовується декілька методів. Одним з поширених є використання стильового параметра float. Хоча він призначений для створення обтікання навколо елемента, з тим же успіхом float встановлює і колонки. Але тут слід врахувати, що при зменшенні вікна браузера до певної критичної ширини колонки перестають розташовуватися горизонтально та перебудовуються одна під одною по вертикалі. З цим фактом залишається змиритися або використовувати інші методи формування колонок через шари.

Блокові елементи

Блоковим називається елемент, який відображується на веб-сторінці у вигляді прямокутника. Такий елемент займає всю доступну ширину, висота елемента визначається його вмістом, і він завжди починається з нового рядка. До блокових елементів відносяться контейнери <DIV>, <H1>, <P> та ін.

Допускається вкладати один блоковий елемент всередину іншого, а також розміщувати всередині них вбудовані елементи (<SPAN>, наприклад). Заборонено додавати всередину вбудованих елементів блокові (рис. 3.2).

У даному прикладі параграф (тег <P>) вставляється всередину контейнера <DIV>, а посилання (тег <A>) – у заголовок <H2>. До речі, помилкою буде вчинити навпаки: додати <H2> до контейнеру <A> (<a href="link1.html"> <h2> Ut wisi </ h2> </ a>), оскільки тег <A> не відноситься до блокових елементів.

Ширина блокових елементів

За замовчуванням, ширина блоку обчислюється автоматично та займає весь доступний простір. Слід обумовити, що під цим розуміється. Наприклад, якщо тег <DIV> в коді документа присутній один, то він займає всю вільну ширину вікна браузера та ширина блоку дорівнюватиме 100 %. Варто помістити один тег <DIV> всередину іншого, як ширина внутрішнього тегу починає обчислюватися щодо його батька, тобто зовнішнього контейнера.

```
<!DOCTYPE>
        <html>
          <head>
             <title>Блочні елементи</title>
             <meta charset="utf-8">
          </head>
          <body>
             <div>
        <p>Lorem ipsum dolor sit amet...</p>
          </div>
             <h2>
        <a href="link1.html">Ut wisi enim ad minim veniam</a>
           </h2>
          </body>
</html>
```

Рис. 3.2. **Використання блокових елементів**

Деякі браузери досить вільно трактують поняття ширини, хоча в специфікації CSS чітко вказано, що ширина складається з суми наступних параметрів: ширини самого блоку (**width**), відступів (**margin**), полів (**padding**) і меж (**border**). На рис. 3.3а показано створення шару, в якому присутні всі ці компоненти.

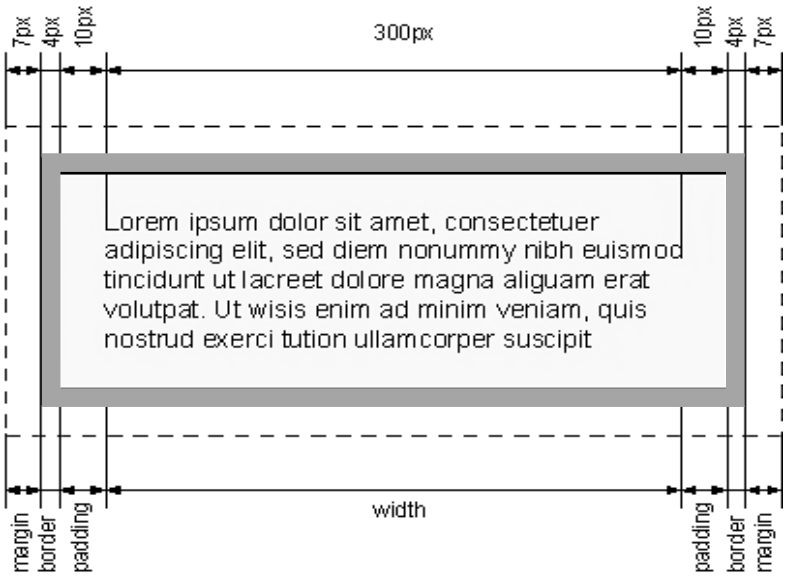
У даному прикладі отримується шар шириною 342 пікселя. На рис. 3.3 б показано, з чого складається ширина шару.

Якщо !DOCTYPE в коді не вказаний, браузер Internet Explorer за ширину всього блоку приймає значення параметра **width.**

Слід розглянути ще один приклад, пов'язаний з шириною. За замовчуванням, ширина шару задається як **auto.** Це дозволяє вписувати шар у вікно браузера, не беручи в розрахунок значення встановлених полів. Якщо змінити ширину на 100 %, то при додаванні значення відступів, полів або меж неминуче з'явиться горизонтальна смуга прокрутки.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Ширина</title>
    <meta charset="utf-8">
    <style type="text/css">
      DIV {
       width: 300px; /* Ширина шару */
       margin: 7px; /* Значення відступів */
       padding: 10px; /* Поля навколо тексту */
       border: 4px solid black;/*Параметри кордону*/
       background: #fc0; /* Колір фону */
      }
    </style>
  </head>
  <body>
    <div>Lorem ipsum dolor sit amet...</div>
  </body>
</html>
```

а



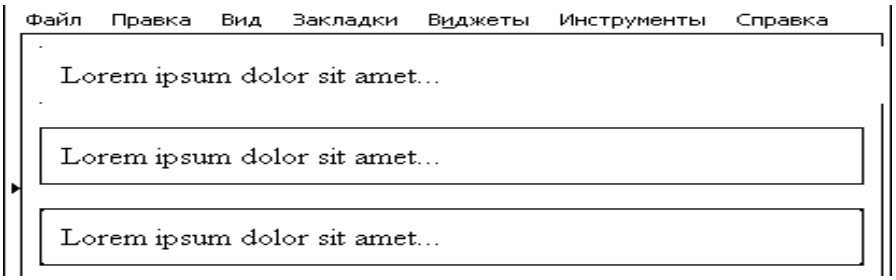
б

Рис. 3.3. **Ширина шару**

Для отримання універсального результату існує кілька підходів. На рис. 3.4 продемонстровано створення трьох шарів, ширина яких визначається у відсотках.

```
<!DOCTYPE HTML>
        <html>
          <head>
            <title>Ширина</title>
            <meta charset="utf-8">
            <style type="text/css">
               #layer1 {
                width: 100%; /* Ширина першого шару */
                padding: 10px; /* Поля навколо тексту */
                background: #fc0; /* Колір фону */
               }
               #layer2 {
                width: 100%; /* Ширина другого шару */
                background: #cc0; /* Колір фону */
               }
               #layer2 P {
                padding: 10px; /* Поля навколо параграфа */
               }
               #layer3 {
              background: #3ca; /*Колір фону третього шару */
                padding: 10px; /* Поля навколо тексту */
               }
            </style>
          </head>
          <body>
            <div id="layer1">Lorem ipsum dolor sit amet...</div>
            <div id="layer2"><p>Lorem ipsum dolor sit amet...</p> </div>
            <div id="layer3">Lorem ipsum dolor sit amet...</div>
          </body>
</html>
```

а



б

Рис. 3.4. **Варіанти установки ширини шару у відсотках**

Ширина першого шару в даному прикладі (layer1) встановлена як 100 %, що призводить до відображення горизонтальної смуги прокрутки. Для другого шару (layer2) ширина також задана 100 %, але поля визначаються для внутрішнього параграфа (тег <P>). За рахунок цього ширина шару в усіх браузерах буде однаковою. До третього шару (layer3) взагалі не застосовується параметр width, тому він визначається за замовчуванням – auto. У такому випадку шар буде займати всю ширину вікна браузера без будь-яких горизонтальних смуг.

Спосіб установки ширини залежить від застосовуваного макета та вибору розробника, але в кожному разі потрібно враховувати особливості блочних елементів і створювати універсальний код.

Висота

Висота блокових елементів встановлюється за аналогією до ширини. Тобто браузер Internet Explorer (а також Opera) за висоту шару приймає значення параметра height, а Firefox додає до нього ще значення параметрів margin, padding і border. Якщо висота шару не встановлена явно, то вона обчислюється автоматично, виходячи з обсягу вмісту.

Колір фону

Колір фону елемента зручніше встановлювати через універсальний параметр background. Фоном при цьому заливається область, яка визначається значеннями атрибутів width, height і padding (рис. 3.5).

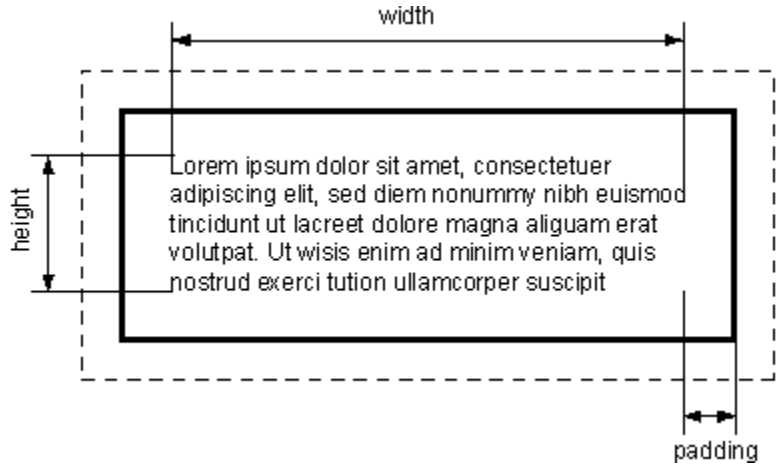
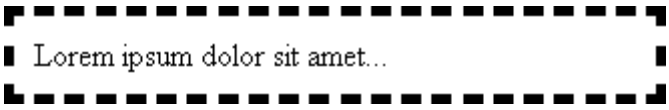


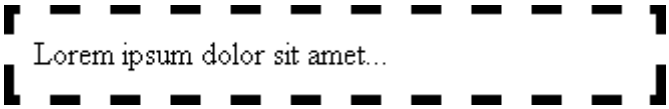
Рис. 3.5. **Область шару, яка заповнюється фоновим кольором**

Межі

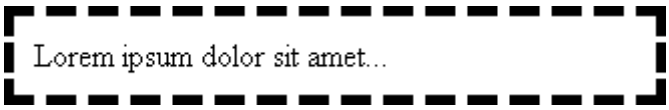
Через відмінність у підходах до браузерів під час формування блокових елементів спостерігається відмінність при відображенні кордонів. Браузер Internet Explorer проводить рамку всередині блоку, а Firefox – іззовні. Але якщо використовувати фонову заливку, то ситуація зміниться (рис. 3.6), тому що Firefox (Opera) колір фону встановлює по зовнішньому краю кордону, а Internet Explorer – по внутрішньому.



А. Internet Explorer



Б. Firefox



В. Opera

Рис. 3.6. **Відображення рамки елемента в різних браузерах**

Відмінності в підході до браузерів при малюванні кордонів помітні тільки на кольоровому фоні та пунктирних лініях. Для суцільної рамки вид шару в браузерах буде практично однаковим.

Вбудовані елементи

Вбудованими називаються такі елементи веб-сторінки, які є безпосередньою частиною іншого елемента, наприклад текстового абзацу. До вбудованих елементів належать теги <SPAN>, <A>, <Q>, <CODE> та ін. В основному вони використовуються для зміни вигляду тексту або його логічного виділення.

Відмінність між блоковими та вбудованими елементами наступна:

вбудовані елементи можуть містити тільки дані або інші вбудовані елементи, а в блокові допустимо вкладати інші блокові елементи, вбудовані елементи, а також дані. Іншими словами, вбудовані елементи не можуть зберігати блокові елементи;

блокові елементи завжди починаються з нового рядка, а вбудовані в такий спосіб не акцентуються;

блокові елементи займають всю доступну ширину, наприклад, вікна браузера, а ширина вбудованих елементів дорівнює їх вмісту плюс значення відступів, полів і меж.

Вбудовані елементи можна перетворювати на блокові за допомогою властивості **display** і його значення **block**. Також можлива і зворотня дія через аргумент **inline**. Основні можливі значення властивості **display** наведено в табл. 3.1; повний список можна знайти за адресою (http://htmlbook.ru/css/display.html).

Таблиця 3.1

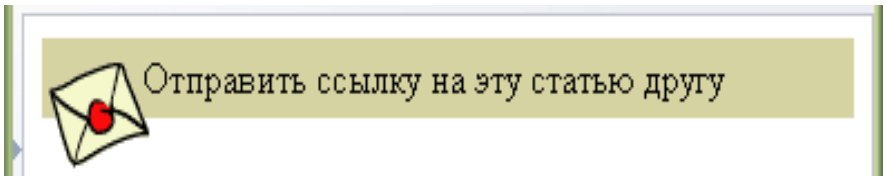
Основні можливі значення властивості display

Аргумент	Опис
block	Елемент показується як блоковий. Застосування цього значення для вбудованих елементів, наприклад тегу **<SPAN>**, змушує його вести подібно до блоків – відбувається перенесення рядків на початку і в кінці вмісту
inline	Елемент відображається як вбудований. Використання блокових тегів – таких, як **<DIV>** і **<P>**, автоматично створює перенос і показує вміст цих тегів з нового рядка. Аргумент inline скасовує цю особливість, тому вміст блокових елементів починається з того місця, де закінчився попередній елемент
inline-block	Це значення генерує блоковий елемент, який обтікається іншими елементами веб-сторінки подібно до вбудованого елемента. Фактично такий елемент за своєю дією схожий на вбудовані елементи (на кшталт тегу **<IMG>**). При цьому його внутрішня частина форматується як блоковий елемент, а сам елемент – як вбудований
list-item	Елемент виводиться як блоковий, і додається маркер списку
none	Тимчасово видаляє елемент з документа. Займане ним місце не резервується, і веб-сторінка формується так, немов елемента нема. Змінити значення параметра та зробити його знову видимим можна за допомогою скриптів, звертаючись до властивостей через об'єктну модель. У цьому випадку відбувається переформатування даних на сторінці з урахуванням знов доданого елемента

Вбудовані елементи застосовуються не тільки для управління виглядом тексту, але й при верстанні веб-сторінок, наприклад для зміни розташування шарів. На рис 3.7 показано, як накладати малюнок поверх блоку з текстом.

```
<html>
        <head>
          <meta charset="utf-8">
          <title> Вбудовані елементи </title>
          <style type="text/css">

            .send {background: #d6d3a2; /* Колір фону */
                   padding: 5px; /*Поля навколо тексту */
                   padding-left: 45px; /*Відступ зліва*/}
            .pic { position: relative; /* Відносне позиціонування */
                   top: -20px; /* Зміщуємо шар вгору */
                   left: 3px; /* Зрушуємо шар вправо */}
          </style>
        </head>
        <body>
          <div class="send"> Надіслати посилання на цю статтю другові</div>

          <div><span class="pic"><img src="mail.gif" width="45" height="38"
alt=""></span></div>

        </body>
</html>
```

а



б

Рис. 3.7. **Розташування малюнка щодо тексту**

За рахунок того, що тег <SPAN> починається не з нового рядка, малюнок і текст в даному прикладі знаходяться на одній лінії. Тому

вертикального відступу під текстом, як у попередньому прикладі, тут нема, але з'явився відступ зверху.

Змінні елементи

Змінними слід називати такі елементи, які обтікаються по контуру іншими об'єктами веб-сторінки, наприклад текстом. У даному випадку необхідно визначитися щодо використання термінів. Зазвичай "плаваючий елемент" є сленговим, правильніше говорити "обтічні елементи". Але термін "змінний елемент" давно вже прижився, тому буде використовуватись надалі.

Змінні елементи досить активно застосовуються в процесі верстання та в основному слугують для втілення таких завдань:

обтікання малюнків текстом;

створення врізок;

розташування шарів по горизонталі (додавання колонок).

Ці операції виконує стильовий параметр **float**, а допомагають йому в цьому інші властивості. Хоча спочатку **float** не планувався на універсальну роль, але життєві реалії розставили все по своїх місцях.

Обтікання малюнків текстом

Існують різні способи, як об'єднувати текст і малюнки до нього. Зазвичай малюнок вирівнюється по лівому або правому краю, а текст обтікає його по контуру.

Створення вставок

Вставкою називається блок з малюнками та текстом, який вбудовується в основний текст. Вставка зазвичай розташовується по лівому або правому краю текстового блоку, а основний текст обтікає її з інших боків.

Щоб вставка виділялася в тексті, у неї зазвичай встановлюють фоновий колір і додають рамку. При створенні вставок слід обов'язково вказувати їх ширину за допомогою параметра **width**, інакше розмір шару виявиться набагато ширшим, ніж це потрібно.

Розташування шарів по горизонталі

За замовчуванням, шари шикуються по вертикалі один під іншим, але за допомогою властивості **float** їх можна змусити розташуватися

поруч по горизонталі. При цьому потрібно встановити ширину шарів і задати для них атрибут **float**.

Оскільки для другого шару також застосовується обтікання, то доданий нижче текст буде розміщений праворуч від шару. Уникнути цього допоможе параметр **clear**, він скасовує дію властивості **float**.

Створення колонок за допомогою **float** має ряд особливостей. Перша, як уже згадувалось, полягає в тому, що після змінних елементів потрібно додавати елемент з властивістю **clear**, який вимикає обтікання. Це необхідно в тому випадку, якщо передбачається використовувати нижні шари. Друга особливість пов'язана з поданням змінних шарів. Якщо вікно браузера зменшити до певної межі, то шари переміщуються по вертикалі.

Вирівнювання шару по центру

Основна відмінність веб-сторінки від аркуша паперу полягає в їх розмірах. Якщо аркуш має задану фіксовану ширину та висоту, то відносно веб-сайту такого сказати не можна. Веб-документ відображується у вікні браузера та може змінювати свої розміри залежно від налаштувань операційної системи, типу монітора, встановленої роздільної здатності тощо. Використання вирівнювання дозволяє проігнорувати вказану особливість і розташувати елемент по краю вікна або по його центру.

Коли мова йде про використання шарів, то для вирівнювання є кілька способів: за допомогою відступів, через позиціонування.

Використання відступів

Якщо додати відступ до шару зліва за допомогою параметра **margin-left**, то візуально шар зміститься на вказане значення вправо. Знаючи ширину шару, його можна змістити так, щоб шар розташовувався по центру веб-сторінки. Для чого від 100 % складових загальної доступної ширини треба відняти ширину шару у відсотках і отримане значення розділити надвоє. Результат і буде значенням параметра **margin-left** (рис. 3.8).

Як варіант, можна не вказувати ширину, а регулювати її за допомогою однакових значень відступів зліва та справа (**margin-left** і **margin-right**).

Наступний спосіб – більш універсальний і вже не залежить від того, які одиниці вимірювання використовуються для встановлювання ширини. Для цього потрібно задати для шару відступ зліва та справа рівним до **auto** через стильові атрибути **margin-left** і **margin-right** або універсальну властивість **margin** (рис. 3.9).

```
...<style type="text/css">
            #centerLayer {
             margin-left: 30%; /* Відступ зліва */
             width: 40%; /* Ширина шару */
             background: #fc0; /* Колір фону */
             padding: 10px;/* Поля навколо тексту */
            }
          </style>
        </head>
        <body>
          <div id="centerLayer">
         Lorem ipsum dolor sit amet, consectetuer dipiscing elit, sed diem nonummy
         nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.
</div>...
```

Рис. 3.8. **Вирівнювання шару по центру за допомогою параметра *margin-left***

```
...<style type="text/css">
            #centerLayer {
             width: 400px; /* Ширина шару в пікселях */
             margin: 0 auto; /* Відступ зліва та справа */
      background: #fc0; /* Колір фону */
             padding: 10px; /* Поля навколо тексту */
            }
          </style>
        </head>
        <body>
          <div id="centerLayer">
            Lorem ipsum dolor sit amet, consectetuer dipiscing elit, sed diem nonummy nibh
euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.
</div>...
```

Рис. 3.9. **Вирівнювання шару по центру за допомогою значення *auto***

У даному прикладі ширина шару встановлюється у 400 пікселів і вирівнюється по центру за допомогою значення **0 auto** параметра **margin**. Перший аргумент встановлює нульовий відступ водночас зверху та знизу від шару, а другий аргумент вирівнює шар по центру горизонталі вікна браузера.

Абсолютне позиціонування шару

Під час абсолютного позиціонування координати шару обчислюються щодо лівого верхнього кута вікна батьківського елементу або браузера, якщо батька немає. Шар, заданий з абсолютним позиціонуванням, може розташовуватися під основним текстом або, навпаки, поверх нього. Розташування визначається за допомогою стильової атрибута **z index** і дозволяє гнучко управляти розташуванням шару за умовною **z** осі. У такий спосіб зручно виводити на веб-сторінці різноманітні підказки, спливаючі вікна, рекламу або змінні меню.

Спочатку слід вказати ширину та висоту шару за допомогою параметрів **width** і **height**. Розміри можна задавати в пікселях, відсотках або інших одиницях. Ширину, наприклад, можна визначити у відсотках, а висоту – в пікселях. Через цю особливість пропонований метод розміщення по центру є найбільш універсальним.

Наступний крок – задати абсолютне позиціонування шару через аргумент **position**: **absolute**. Розташування шару слід визначити як 50 % по горизонталі та вертикалі за допомогою властивостей **left** і **top**. Ці значення залишаються незмінними незалежно від використовуваних одиниць виміру.

Оскільки координати шару визначаються від його лівого верхнього кута, для точного вирівнювання слід додати параметри **margin-left** і **margin-top** з негативними значеннями. Їх величина повинна дорівнювати половині ширини шару (для **margin-left**) і висоти (для **margin-top**).

Щоб висота шару не змінювалася через його контенту, включається параметр **overflow**: **auto**, який додає смуги прокрутки, якщо в них виникне потреба. Висота при цьому залишається незмінною (рис. 3.10).

```
...
            <style type="text/css">
              #centerLayer {
               position: absolute; //абсолютне позиціонування
               width: 400px; /* Ширина шару в пікселях */
               height: 300px; /* Висота шару в пікселях */
               left: 50%; /* Розташування шару від лівого краю */
               top: 50%; // Розташування шару від верхнього краю
               margin-left: -200px; /* Відступ зліва */
               margin-top: -150px; /* Відступ зверху */
               background: #fc0; /* Колір фону */
               border: solid 1px black; /* Параметри рамки навколо */
               padding: 10px; /* Поля навколо тексту */
               overflow: auto; /* Додавання смуги прокручування */
              }
            </style>
          </head>

          <body>
            <div id="centerLayer">
               Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem
nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat
volutpat.
            </div>
...
```

Рис. 3.10. **Вирівнювання шару по центру за допомогою абсолютного позиціонування**

Ширина та висота шару безпосередньо пов'язані з відступами зліва та зверху. Якщо потрібно встановити значення одного з параметрів у відсотках, відповідно, зміниться і запис іншого параметра. Зазначена особливість дозволяє застосовувати будь-які одиниці вимірювання, а не обмежуватися тільки однією формою запису, що робить код відповідним практично для всіх випадків.

У якості довідкової інформації в табл. 3.2 наведено опис видів позиціонування шарів.

Таблиця 3.2

Види позиціонування шарів

Аргумент	Опис
absolute	Вказує, що елемент абсолютно позиціонується. У цьому випадку він не існує в звичайному потоці документа подібно до інших елементів, які відображуються на веб-сторінці, ніби абсолютно позиціонованого об'єкта немає. Розташування елемента задається атрибутами **left**, **top**, **right** і **bottom** щодо краю вікна браузера
fixed	За своїми властивостями це значення аналогічно аргументу **absolute**, але, на відміну від нього, прив'язується до зазначеної параметрами **left**, **top**, **right** і **bottom** точці на екрані та не змінює свого положення навіть під час прокручування веб-сторінки
relative	Розташування елемента встановлюється щодо його вихідного місця. Додавання атрибутів **left**, **top**, **right** і **bottom** змінює позицію елемента та зрушує його в той або інший бік від первісного розташування залежно від застосовуваного параметра
static	Елементи відображуються як звичайно. Використання параметрів **left**, **top**, **right** і **bottom** не призводить до якихось результатів

Гумовий дизайн. Двоколонний макет

"Гумовим дизайном" називають структуру веб-сторінки, яка автоматично пристосовується під певну ширину вікна браузера користувача. Такий макет має кілька переваг:

використовується весь доступний простір браузера;

у користувачів з "маленькими" моніторами не виникає горизонтальних смуг прокрутки;

користувачі "великих" моніторів можуть порадіти, що ефективно використовується вся площа екрану, а отже, гроші на монітор були витрачені не даремно;

веб-сторінки зручно друкуються на папері будь-якого формату.

Серед недоліків можна відзначити "прискіпливість" різних браузерів до "гумового" макета і значну ймовірність появи помилок, у зв'язку з чим підвищується складність верстання веб-сторінок.

Для верстання веб-сторінок найбільш популярним є двоколонний макет, при цьому одна колонка містить набір посилань для навігації по сайту, а друга, більш широка – контент. Утім, хоча така схема і є усталеною традицією, це не означає, що її обов'язково слід дотримуватися. Використання всієї ширини вікна дозволяє більш

ефективно задіяти площу веб-сторінки. Отже можна додати три та навіть чотири колонки. Це залежить виключно від наявного обсягу інформації. Враховується і зворотний ефект – чим більше матеріалу, тим більше розсіюється увага відвідувача, і йому складніше стає орієнтуватися на сайті.

Для створення двоколонного макета застосовуються два способи. Перший підхід використовує параметр **float**, що дозволяє зістиковувати один шар з іншим по горизонталі, а другий заснований на завданні розташування шарів через позиціонування.

Використання змінних елементів

Термін "змінний елемент" не дуже вдалий, проте, часом зустрічається в літературі. Так називають елемент, обтічний з різних боків текстом або іншими об'єктами веб-сторінки. Параметр **float**, доданий до шару, визначає, по якому боці буде вирівнюватися шар, при цьому інші елементи будуть обтікати його з інших сторін.

Доцільно розглянути варіант, коли ліва колонка має певний розмір, а ширина правої колонки встановлюється автоматично, виходячи із ширини вікна браузера. При цьому ширина лівої колонки може задаватися в пікселях або відсотках. У табл. 3.3 наведені основні стильові параметри для формування двох колонок.

Таблиця 3.3

Приклади гумового блокового верстання із заданою шириною лівої колонки

Для лівого шару шириною 20 %	
Шар 1 float: left width: 20 %	Шар 2 margin-left: 21 %
Для лівого шару шириною 200 px	
Шар 1 float: left width: 200 px	Шар 2 margin-left: 210 px

Для першого шару потрібно всього два параметри: **float** – змушує другий шар розташовуватися поруч по горизонталі з першим шаром і

width, який встановлює ширину колонки. Друга колонка буде займати все вільне місце, тому для неї атрибут **width** не потрібний.

Правий шар характеризується лише одним параметром – **margin-left**. Він зміщує лівий край колонки на ширину лівого шару та задає відступ між колонками. Тому величина цієї властивості в табл. 3.3 вказана як 21 %, де 20 % – ширина першого шару, а на 1 % припадає відстань між колонками. У разі задавання ширини однієї з колонок в пік - селях код залишиться попереднім, але поміняються одиниці виміру.

Для формування колонки заданої ширини праворуч, а не ліворуч, код несуттєво модифікується. У табл. 3.4 показані стильові параметри, які потрібні для цього випадку.

Таблиця 3.4

Приклади блокового гумового верстання з заданою шириною правої колонки

Для лівого шару шириною 20%	
Шар 1 float: right width: 20 %	Шар 2 margin-right: 21 %
Для лівого шару шириною 200px	
Шар 1 float: right width: 200 px	Шар 2 margin-right: 210 px

Розташування шарів в коді залишається попереднім, але значення атрибута **float** змінюється на **right**, а параметр відступу – на **margin-right.**

Застосування позиціонування

При формуванні двоколонного макета ліва або права колонка встановлюється в задане місце через абсолютне позиціонування, а сусідня колонка звільняє для неї місце за рахунок застосування відступів. Для кращого розуміння слід розглянути приклад, коли ліва колонка має задану ширину 200 пікселів, а її розташування визначається від лівого верхнього кута вікна браузера. Стиль для таких колонок наведено на рис. 3.11.

Значення **absolute** властивості **position** дозволяє задавати розташування шару щодо краю вікна браузера незалежно від наявності

та розташування інших шарів. Самі координати встановлюються за допомогою параметрів **left, top, right** і **bottom**, які, відповідно, визначають відстань від лівого, верхнього, правого та нижнього країв вікна. Щоб ліва та права колонки не накладалися одна на іншу, слід додати відступ зліва (**margin-left**) для шару **rightcol**, як показано в даному прикладі. Значення цього відступу включає відстань від лівого краю (параметр **left**) і ширину лівої колонки (**width**) плюс дистанція між колонками.

```
...
        #leftcol { /* Ліва колонка */
          position: absolute; /* Абсолютне позиціонування */
          width: 200px; /* Ширина шару */
          left: 0; /* Розташування від лівого краю вікна */
        top: 20px; /* Розташування від верхнього краю вікна */
          background: #fc0; /* Колір фону лівої колонки */
        }

        #rightcol { /* Права колонка */
          margin-left: 210px; /* Відступ зліва */
          background: #ccc; /* Колір фону правої колонки */
        }
...
```

Рис. 3.11. **Стилі для гумового верстання за допомогою позиціонування шарів**

Приклад вирішення завдання

Як завдання обрано "шапку" модульної сітки, яка присутня в декількох варіантах завдання (рис. 3.12).

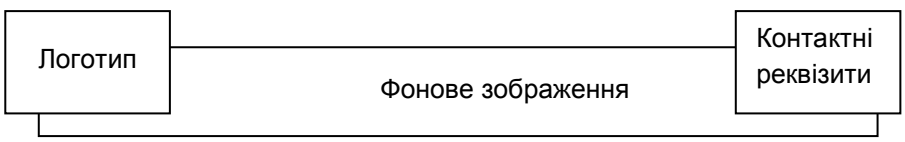


Рис. 3.12. **Модульна сітка прикладу завдання**

Для реалізації даного завдання можна скористатися двома тегами <DIV> (і), які вкладені в ще один тег <DIV id="header"> (рис. 3.13).

```
<div id="header">
  <div id="logo">Логотип</div>
  <div id="contact"> Контактні реквізити </div>
</div>
```

Рис. 3.13. **Етап 1 – розмітка документа**

Для наочності встановлюється рамка для головного тегу <DIV id="header"> і заливається фон вкладених тегів <DIV> різними кольорами (рис. 3.14).

```
<style type="text/css">
    div#header{
      border: green 2px dashed;
    }
    div#logo{
      background:yellow;
    }
    div#contact{
       background:blue;
    }
</style>
```

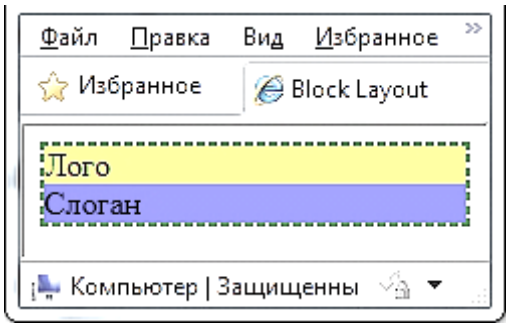


Рис. 3.14. **Етап 2 – додавання наочності блокам**

Наступним етапом встановлюються розміри внутрішніх блоків (наприклад: 100 х 100 px для кожного). Результат наведено на рис. 3.15.

```
div#logo{
      background:yellow;
      width:100px;
      height:100px;
    }
    div#contact{
      background:blue;
      width:100px;
      height:100px;
}
```

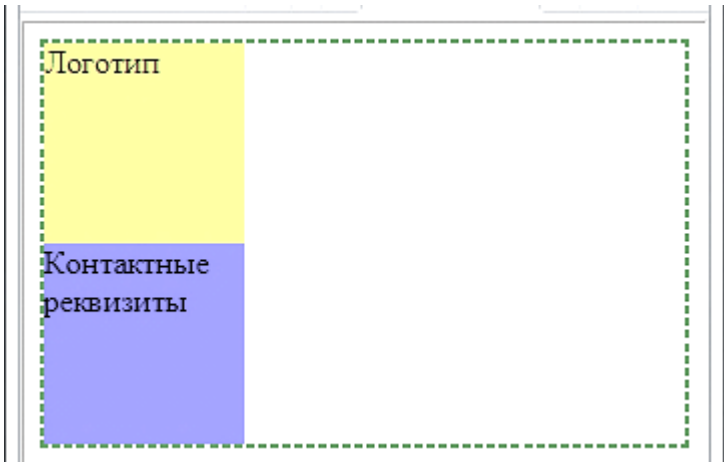


Рис. 3.15. **Етап 3 – встановлення розмірів**

Наступним завданням буде розміщення блоків на одній лінії (в одному рядку). Для цього можна скористатися властивістю **float**, яка задає обтікання тексту ліворуч, праворуч або навколо поточного (того, до якого застосовується це властивість). У даному прикладі потрібно задати обтікання для першого блоку, щоб другий розташовувався праворуч від нього. Для цього потрібно встановити властивість **float** в значення **left** (значення вказує розташування елемента, який будуть обтікати сусідні). Але обтікати блок будуть тільки вбудовані елементи, а блокові (у даному випадку <DIV>) будуть ніби під ним (з меншим значенням координати **Z** – властивість **Z-index**). Тому краще одразу змістити другий блок до правого краю батьківського. Це можна зробити, задавши лівий зовнішній відступ другого блоку (**margin-left**) в значенні **auto**, за якого відступ буде "займати" всю доступну ширину. Результат наведено на рис. 3.16.

```
div#logo{
        background-color:yellow;
        width:100px;
        height:100px;
        float:left;
      }
      div#contact{
        background-color:blue;
        width:100px;
        height:100px;
        margin-left:auto;
}
```

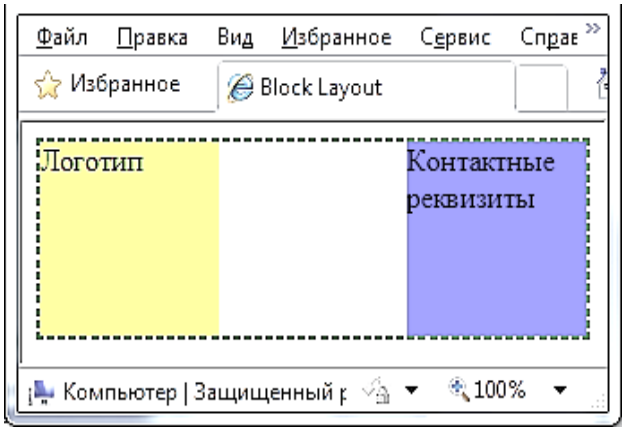


Рис. 3.16. **Етап 4 – установка обтікання та вирівнювання**

Останнім етапом вирішення даного завдання є зміщення лівого блоку вгору та вліво, а правого – вгору та вправо. Для цього цим блокам потрібно задати властивість позиціонування (**position**) відносним (**relative**) і вказати значення зсувів: **left, top, right**. Результат наведено на рис. 3.17.

```
div#logo{
        background-color:yellow;
        width:100px;
        height:100px;
        float:left;
        position:relative;
        top:-5px;
        left:-5px;
      }
      div#contact{
        background-color:blue;
        width:100px;
        height:100px;
        margin-left:auto;
        position:relative;
        top:-5px;
        right:-5px;
}
```

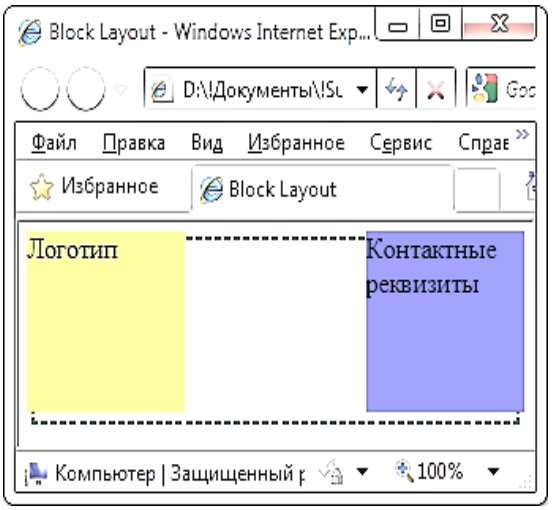


Рис. 3.17. **Етап 5 – встановлення відносного позиціонування і значень зміщень**

3.4. Завдання до роботи

1. Зверстати шаблон сторінок сайту, використовуючи прийоми блокового верстання, згідно з варіантом завдання.

2. У процесі створення шаблону чітко дотримуватися принципу поділу структури (застосовувати HTML) і подання (застосовувати тільки CSS).

3. Тип макета (каркасу) шаблону – гумовий.

4. Поставити мінімальну ширину каркаса й обґрунтувати.

5. Визначити, які блоки (модулі) шаблону будуть гумовими, а які – фіксованими. Вибір обґрунтувати.

6. Усі стилі повинні знаходиться тільки в зовнішньому файлі(ах).

7. Передбачити набори стилів як мінімум для двох пристроїв (*screen* і *printer*).

8. Перевірити шаблон на відповідність до стандартів W3C і на сумісність з найбільш популярними в Україні та світі браузерами.

9. На підставі шаблону створити сайт, що складається мінімум з п’яти сторінок.

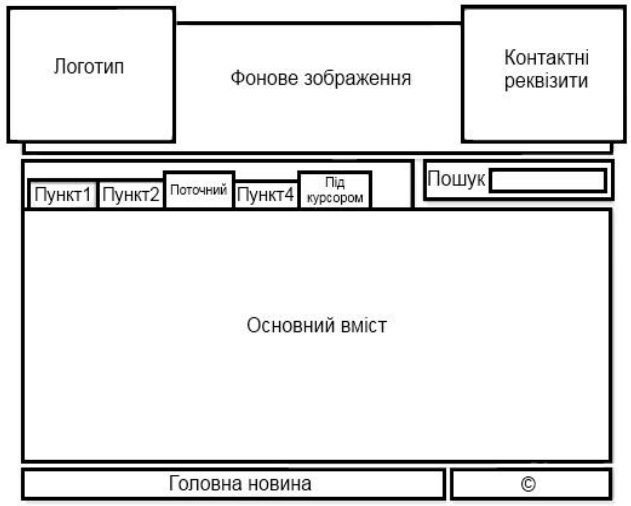
10. Тематика сайта – довільна.

11. Вимоги до структури каталогу сайта відповідають вимогам до лабораторної роботи № 1.

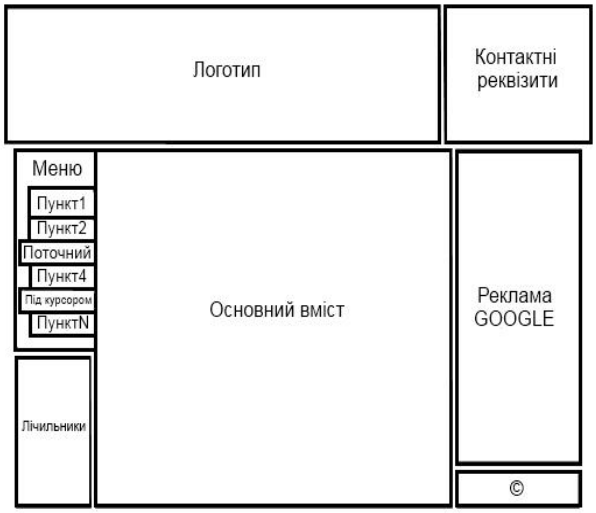
На захисті лабораторної роботи подати сайт, шаблон і продемонструвати відміну стилів для різних пристроїв.

Варіанти завдання

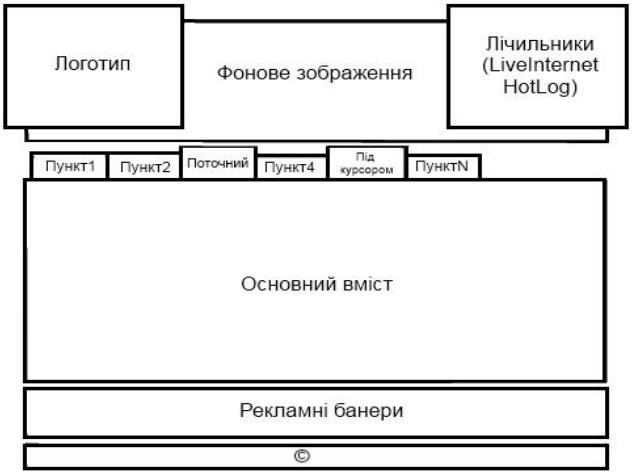
Варіант 1



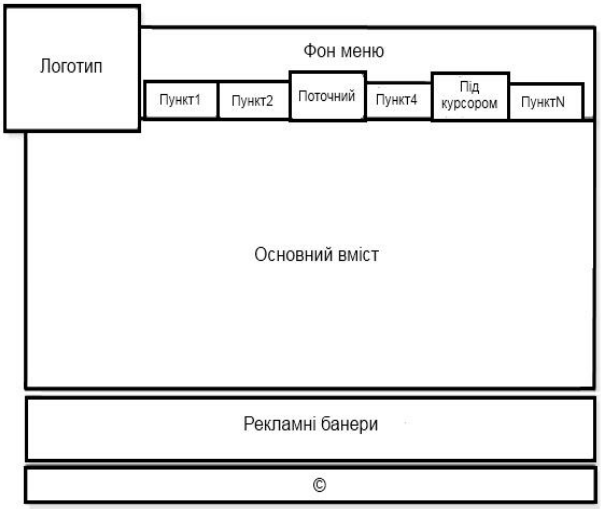
Варіант 2



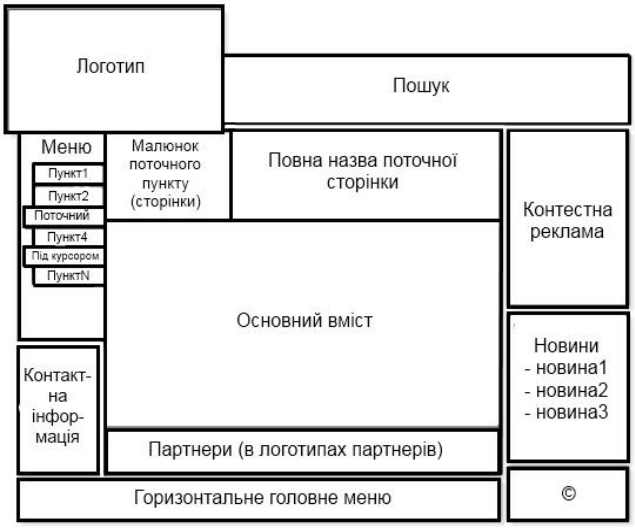
Варіант 3



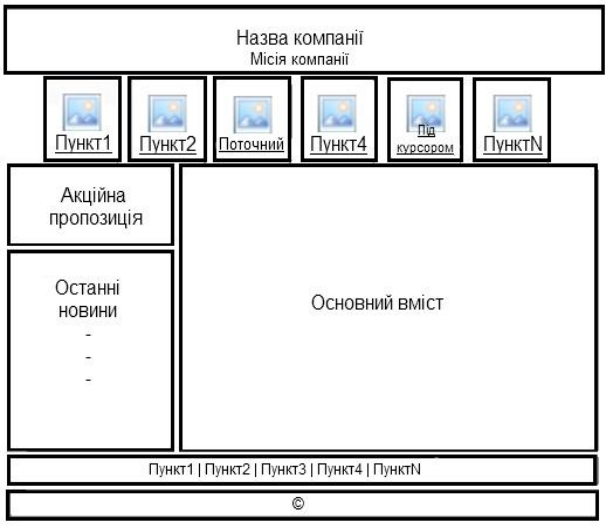
Варіант 4

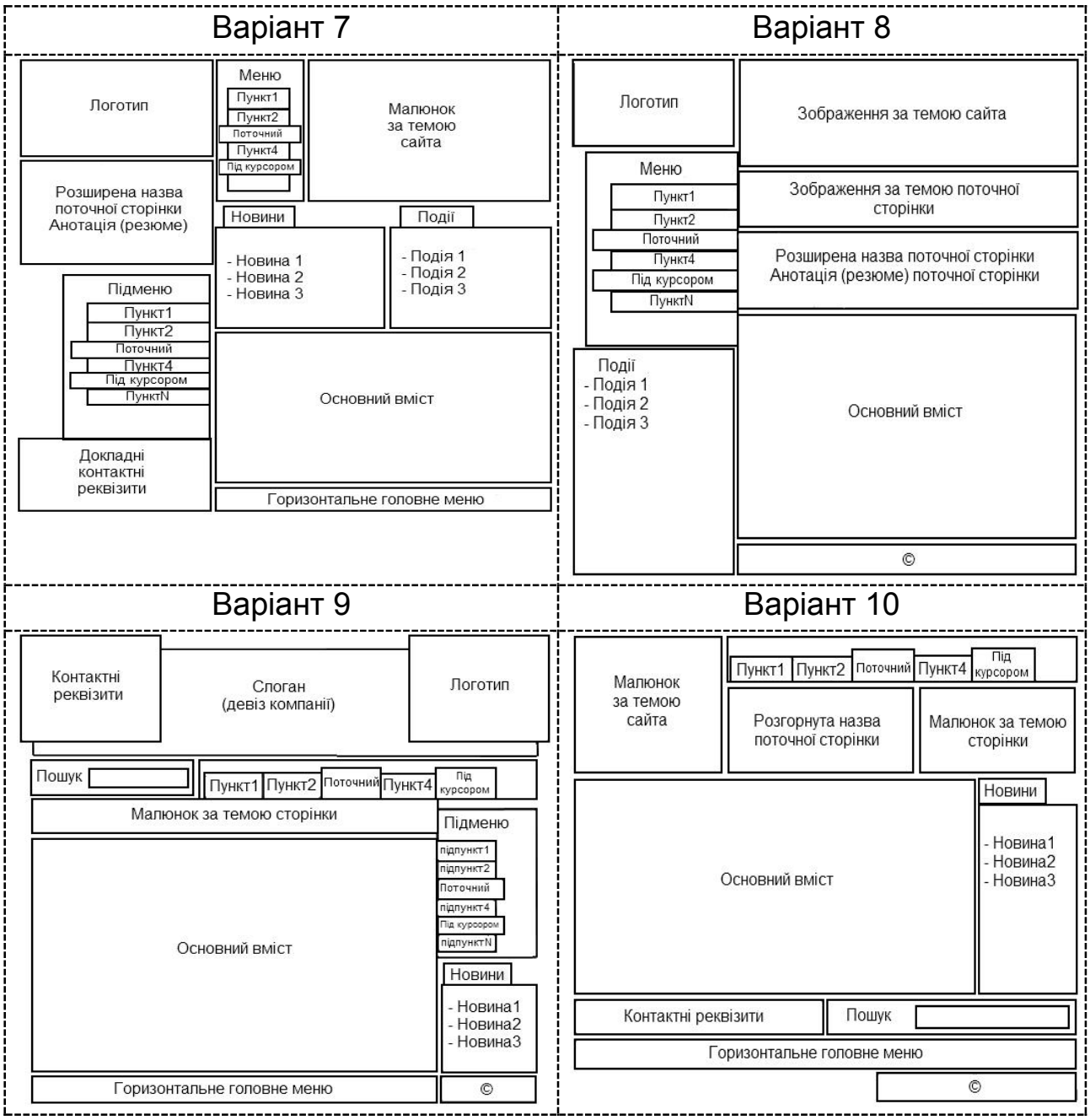


Варіант 5



Варіант 6





3.5. Контрольні запитання

1. Назвіть ознаки блокових та рядкових (вбудованих) елементів HTML.
2. Які основні css-властивості використовують за блокового верстання веб-сторінок?
3. Назвіть способи вирівнювання блоку по центру.
4. Які види позиціонування елементів веб-сторінки вам відомі?
5. Для чого використовується css-властивість **z-index**?
6. Назвіть способи визначення стилів для різних типів пристроїв.
7. У яких випадках ефективно застосовувати директиву **@import**?
8. Чим відрізняються відступи, задані за допомогою css-властивості **padding** від **margin**?

Лабораторна робота № 4
Розроблення динамічних веб-сторінок за допомогою мови JavaScript

4.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення основ мови сценаріїв JavaScript для подальшого застосування під час розробки веб-сайтів.

4.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити основи мови сценаріїв JavaScript для подальшого застосування під час розроблення веб-сайтів.

4.3. Загальні тези лабораторної роботи

JavaScript – це мова управління сценаріями перегляду гіпертекстових сторінок Web на боці клієнта. Якщо бути більш точним, JavaScript – це не тільки мова програмування на боці клієнта. Liveware, предок JavaScript, є засобом підстановок на боці сервера Netscape. Однак найбільшу популярність JavaScript забезпечило програмування на боці клієнта [1 – 4].

Основна ідея JavaScript полягає в можливості зміни значень атрибутів HTML-контейнерів і властивостей середовища відображення в процесі перегляду HTML-сторінки користувачем. При цьому перезаванта-ження сторінки не відбувається. На практиці це виражається в тому, що можна, наприклад, змінити колір фону сторінки або інтегрований в документ малюнок, відкрити нове вікно або видати попередження.

Назва "JavaScript" є зареєстрованим товарним знаком компанії Sun Microsystems. Реалізація мови, здійснена розробниками Microsoft, офіційно називається JScript. Версії JScript сумісні (проте не вповні) з відповідними версіями JavaScript, тобто JavaScript є підмножинною мови JScript. На сьогодні JavaScript повністю займає нішу браузерних мов. На синтаксис JavaScript вплинула мова Java, звідки і пішла назва JavaScript. Як і Java, мова JavaScript є об'єктною. Однак на цьому їх зв'язок закінчується: Java і JavaScript – це різні мови, жодна не є підмножинною іншої.

Стандартизація мови була ініційована компанією Netscape та здійснюється асоціацією ECMA (European Computer Manufacturers Association – Асоціація європейських виробників комп'ютерів). Стандартизована версія має назву ECMAScript і описується стандартом ECMA-262 (доступно в мережі: англійською, російською) [6,13;15].

Розміщення коду JavaScript на HTML-сторінці

Головне запитання будь-якого програміста-початківця: "Як оформити програму та виконати її?" На нього можна відповісти якомога простіше, при цьому не забуваючи про всі способи застосування JavaScript-коду.

По-перше, виконує JavaScript-код браузер. У нього вбудований інтерпретатор JavaScript. Отже, виконання програми залежить від того, коли й як цей інтерпретатор отримує управління. Це, в свою чергу, залежить від функціонального застосування коду. У загальному випадку можна виділити чотири способи функціонального застосування JavaScript:

1) гіпертекстове посилання (схема URL);
2) обробник події (в атрибутах, що відповідають подіям);
3) вставка (контейнер <SCRIPT>).

Нижче вони розглядаються послідовно. У підручниках з JavaScript опис застосування JavaScript зазвичай починають з контейнера <SCRIPT>. Але з позицій розуміння сутності взаємодії JavaScript і HTML це не зовсім правильно, оскільки такий порядок не дає відповіді на ключове питання: як JavaScript-код отримує управління? Інакше кажучи, яким чином викликається та виконується програма, написана на JavaScript і розміщена в HTML-документі?

Спосіб 1: URL-схема "JavaScript"

Основним завданням мови програмування гіпертекстової системи є програмування гіпертекстових переходів. Це означає, що при виборі того або іншого гіпертекстового посилання викликається програма реалізації гіпертекстового переходу. У Web-технології стандартною програмою, створеною при гіпертекстовому переході, є програма завантаження сторінки (тобто при кліку за посиланням завантажується сторінка з зазначеним URL). JavaScript дозволяє змінити стандартну програму на програму користувача. Для того щоб відрізнити стандартний перехід за протоколом HTTP від переходу, програмованого на JavaScript,

розробники мови ввели нову схему URL – JavaScript. Приклади загального застосування даної схеми наведено на рис. 4.1.

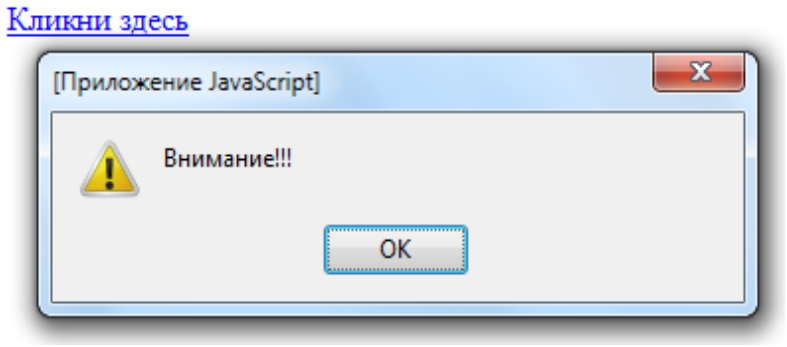
```
<A HREF="JavaScript:код_програми">...</A>
<FORM ACTION="JavaScript:код_програми" ...> ... </FORM>
```

Рис. 4.1. **Приклади застосування URL-схеми JavaScript**

У даному випадку текст "код_програми" позначає програму-обробник на JavaScript, яка викликається під час вибору гіпертекстового посилання в першому випадку та при відправці даних форми (натискання кнопки Submit) – у другому. Наприклад (рис. 4.2), після натискання на гіпертекстове посилання "Натисни тут" з'явиться вікно попередження.

```
<A HREF="JavaScript:alert('Увага!!!');">Клікни тут</A>
```

а) **Код**



б) **результат виконання**

Рис. 4.2. **Приклад використання JavaScript в гіперпосиланні**

Натисканням на кнопку типу **submit** у формі можна заповнити текстове поле цієї ж форми (рис. 4.3).

В URL можна розміщувати складні програми та викликання функцій. Таким чином, у процесі програмування гіпертекстового переходу JavaScript-інтерпретатор отримує управління після того, як користувач "клацнув" по гіпертекстовим посиланням.

```
<FORM METHOD="post" NAME="form"
  ACTION="JavaScript:form.e.value='Натиснути кнопку:
Заповнити';void(0);">
        <INPUT TYPE="text" NAME="e" SIZE="30" VALUE=""><BR>
        <INPUT TYPE="submit" VALUE="Заповнити">
        <INPUT TYPE="reset" VALUE="Очистити">
</FORM>
```

Рис. 4.3. **Заповнення поля після натискання кнопки**

Спосіб 2: обробники подій

Такі програми, як обробники подій, вказуються в атрибутах контейнерів, з якими ці події пов'язані. Наприклад, у разі натискання на кнопку відбувається подія **Click** і, відповідно, викликається обробник цієї події **onClick** (рис. 4.4).

```
<FORM>
        <INPUT TYPE="button" VALUE="Кнопка" onClick="alert('Ви натиснули
кнопку');">
</FORM>
```

Рис. 4.4. **Оброблення події Click**

У момент завершення повного завантаження документа (він пов'язаний з контейнером <BODY>) відбувається подія **Load** і, відповідно, буде викликаний оброблювач цієї події **onLoad** (рис. 4.5).

```
<BODY onLoad="alert('Вітаємо!');">
   ...
</BODY>
```

Рис. 4.5. **Оброблення події Load**

Спосіб 3: вставка (контейнер <SCRIPT>)

Під час розбору документа HTML-парсер передає управління JavaScript-інтерпретатору після того, як зустріне тег початку контейнера <SCRIPT>. Інтерпретатор отримує на виконання весь фрагмент коду

всередині контейнера SCRIPT і повертає управління HTML-парсеру для оброблення тексту сторінки після тегу кінця контейнера </ SCRIPT>.

Поміщати JavaScript-код на HTML-сторінці за допомогою контейнера <SCRIPT> можна двома способами. Перший полягає в написанні тексту коду безпосередньо всередині цього контейнера (рис. 4.6).

```
<SCRIPT>
        var a = 5;
</SCRIPT>
```

Рис. 4.6. **Вставка JavaScript-коду всередині контейнера SCRIPT**

Другий спосіб полягає в тому, щоб винести код JavaScript в окремий файл, наприклад **myscript.js** (розширення може бути будь-яким), і потім включити його в HTML-сторінку, як показано на рис. 4.7. Цей спосіб зручний, коли один тот скрипт планується використовувати на різних HTML-сторінках.

```
<SCRIPT TYPE="text/JavaScript" SRC="myscript.js"></SCRIPT>
```

Рис. 4.7. **Підключення файла з JavaScript-кодом**

За стандартом, атрибут **type** потрібен для вказівки на мову скрипту, але, за замовчуванням, прийнятий JavaScript, тому за відсутності атрибута **type** все буде працювати. Проте валідатор буде "сперечатися", оскільки стандарт вимагає наявності цього атрибута.

За вказівкою атрибута **src** вміст тегу ігнорується. Тобто одночасно підключити зовнішній файл і написати щось усередині тегу можна. Доведеться вставити два різних теги <script>: перший – з **src**, другий – з командами, які будуть виконані після виконання зовнішнього файлу.

Зазвичай JavaScript намагаються відокремити від власне документа. Для цього його поміщають всередину тега HEAD, а в тілі сторінки, по можливості, залишають тільки верстання.

Контейнер SCRIPT виконує дві основні функції:

розміщення коду всередині HTML-документа;

умовна генерація HTML-розмітки на боці браузера.

Перша функція аналогічна декларуванню змінних і функцій, які потім можна буде використовувати як програму переходів, обробників подій і підстановок. Друга – це підстановка результатів виконання JavaScript-коду в момент завантаження або перезавантаження документа.

Розміщення коду всередині HTML-документа

Код можна розмістити як в заголовку документа (всередині контейнера HEAD), так і в тілі документа (всередині контейнера BODY).

Код в заголовку документа розміщується всередині контейнера SCRIPT. У наступному прикладі (рис. 4.8) оголошується функція **time_scroll()** в заголовку документа, а потім вона викликається як обробник події Load в тезі BODY.

```
<HTML>
        <HEAD>
          <TITLE> Розміщення коду всередині HTML-документа </TITLE>
          <SCRIPT>
            function time_scroll()
            {
              var d = new Date();
              window.status = d.getHours()
                    + ':' + d.getMinutes()
                    + ':' + d.getSeconds();
              setTimeout('time_scroll()',1000);
            }
          </SCRIPT>
        </HEAD>
        <BODY onLoad="time_scroll()">
          <H1> Годинники в рядку стану </H1>
        </BODY>
</HTML>
```

Рис. 4.8. **Розміщення коду всередині HTML-документа**

Функція **time_scroll()** викликається по закінченні повного завантаження документа (обробником **onLoad**). Вона заносить поточну дату та час (**new Date**) до змінної **d**. Потім записує поточний час у форматі ГГ: ММ: СС в **window.status**, тим самим він буде відображуватися в полі статусу вікна браузера. Нарешті, вона відкладає (**setTimeout**) повторний

самовиклик на 1 000 мілісекунд (тобто 1 секунду). Таким чином, кожну секунду в полі статусу буде відображуватися новий час.

Умовна генерація HTML-розмітки на боці браузера

Завжди приємно отримувати з сервера сторінку, підбудовану під можливості браузера або, більше того, під користувача. Існує тільки дві можливості генерації таких сторінок: на боці сервера або безпосередньо у клієнта. JavaScript-код виконується на боці клієнта.

Для генерації HTML-розмітки контейнер SCRIPT розміщують у тілі документа, тобто всередині контейнера BODY. Простий приклад – вбудовування в сторінку локального часу – наведено на рис. 4.9.

```
<BODY>
        ...
        <SCRIPT>
           d = new Date();
           document.write(Момент завантаження сторінки: '
           + d.getHours() + ':'
           + d.getMinutes() + ':'
           + d.getSeconds());
        </SCRIPT>
        ...
</BODY>
```

Рис. 4.9. **Умовна генерація HTML-розмітки на боці браузера**

Коментарі в HTML і JavaScript

У програмі на JavaScript можна залишати коментарі, які ігноруються JavaScript-інтерпретатором і слугують поясненням для розробників. Однорядкові коментарі починаються з символів //. Текст, починаючи з цих символів і до кінця рядка, вважається коментарем. Багаторядковий коментар укладається між символами/* і */і може займати кілька рядків.

На відміну від мови розмітки HTML, мова JavaScript є регістро-залежною.

Типи даних і оператори

Як і будь-яка інша мова програмування, JavaScript підтримує вбудовані структури та типи даних. Усе їх різноманіття поділяється на:

літерали;

змінні;

масиви;

функції;

об'єкти.

Вони розрізняються на: вбудовані та визначені програмістом.

Літерали

Літералом називають дані, які використовуються безпосередньо в програмі. При цьому під даними розуміють числа або рядки тексту. Усі вони розглядають в JavaScript як елементарні типи даних. Приклади літералів:

числовий літерал**:** 10

числовий літерал**:** 2.310

числовий літерал**:** 2.3e**+**2

рядковий літерал**:** 'Це рядковий літерал'

рядковий літерал**:** "Це рядковий літерал"

Літерали використовуються в операціях присвоювання значень змінним або в операціях порівняння:

```
var a=10;
var str = 'Рядок';
if(x=='test') alert(x);
```

Оператор присвоювання (змінна = вираз) повертає результат обчислення виразу, тому ніщо не заважає отримане значення надати ще й іншій змінній. Таким чином, послідовність операторів присвоювання виконується справа наліво:

```
result = x = 5 + 7;
```

Два варіанти рядкових літералів необхідні для того, щоб використовувати вкладені рядкові літерали. Якщо в рядковому літералі

потрібно використовувати одинарні лапки, то сам літерал можна укласти в подвійні лапки: "It's cool!". Правильно і зворотне. Але якщо є необхідність використовувати в рядковому літералі обидва види лапок, то найпростіше всіх їх "екранувати" символом зворотної косої межі \, при цьому сам рядок можна укласти в будь-яку пару лапок. Наприклад:

Команда:

```
document.write("It\'s good to say \"Hello\" to someone!");
```

видасть:

It's good to say "Hello" to someone!

Крім рядкових літералів (послідовностей символів, укладених в лапки) є ще рядкові об'єкти; вони створюються конструктором: **var s = new String()**. Цей об'єкт має багато методів. Слід розуміти, що рядковий літерал та рядковий об'єкт – далеко не те ж саме. Але найчастіше на це не зважають, тому що при застосуванні до рядкових ЛІТЕРАЛ – методів рядкових об'єктів відбувається перетворення перших в останні.

Наприклад, можна спочатку присвоїти *var s = 'abra-kadabra'*, а потім застосувати метод: *var m = s.split ('b')*, який неявно перетворює рядковий літерал s у рядковий об'єкт і далі розбиває рядок в тих місцях, де зустрічається підрядок *'b'*, повертаючи масив рядків-шматів: масив m складатиметься з рядків *'a'*, *'ra-kada'* і *'ra'*.

Змінні

Змінна – це область пам'яті, що має своє ім'я та зберігає певні дані. Змінні в JavaScript оголошуються за допомогою оператора var, при цьому можна надавати чи не надавати їм початкові значення:

```
var k;
var h='Привіт!';
```

Можна оголошувати одразу кілька змінних в одному операторі **var** (тим самим зменшуючи розмір коду), але тоді їх треба писати через кому. При цьому теж можна надавати чи не надавати початкові значення:

```
var k, h='Привіт!';
var t=37, e=2.71828;
```

Тип змінної визначається за присвоєним їй **значенням.** Мова JavaScript – **слабо типізована**: у різних частинах програми можна привласнювати тій самій змінній значення різних типів, і інтерпретатор буде "на льоту" змінювати тип змінної. Дізнатися тип змінної можна за допомогою оператора **typeof():**

```
var i=5;          alert(typeof(i));
i= new Array();   alert(typeof(i));
i= 3.14;          alert(typeof(i));
i= 'Привіт!';     alert(typeof(i));
i= window.open(); alert(typeof(i));
```

Змінна, оголошена оператором **var** поза функціями, є глобальною – її "видно" повсюди в скрипті. Змінна, оголошена оператором **var** всередині функції, є локальною – її видно тільки в межах цієї функції.

Наприклад, у наступному фрагменті ніщо не буде виведене на екран, незважаючи на те, що звернення до змінної **k** відбувається після опису функції і навіть після її виклику:

```
function f()
{ var k=5; }

f(); alert(k);
```

Якщо наявна глобальна змінна **k**, а всередині функції оголошується локальна змінна з тим же ім'ям (оператором **var k**), то це буде інша змінна, і зміна її значення усередині функції не вплине на значення глобальної змінної з тим же ім'ям.

Те ж стосується й аргументів при описі функцій (але перед ними не потрібно ставити **var**): якщо є глобальна змінна **k** і описується функція **function f (k) {...}**, то змінна **k** всередині **{...}** не пов'язана з одноіменною глобальною змінною. У цьому плані JavaScript не відрізняється від інших мов програмування.

Повідомляти змінні можна і без оператора **var**, просто привласнюючи змінній початкове значення. Так часто роблять зі змінними циклів. У наступному прикладі, навіть якщо змінна не була визнана раніше, все буде працювати коректно:

```
for(i=0; i<8; i++) {... }
```

Однак опускати **var** не рекомендується. По-перше, це порушує ясність коду: якщо написано i = 5, то незрозуміло, вводиться тут нова змінна чи змінюється значення існуючої. По-друге, це часто призводить до неправильної роботи програми.

Поза функцій, оголошення змінної без оператора **var** рівнозначне оголошенню з оператором **var** – в обох випадках змінна буде глобальною. Усередині ж функції оголошення змінної без оператора **var** робить змінну глобальною (а не локальною, як можна було б припустити). Отже, її значення можуть "бачити" і змінювати інші функції або оператори поза цією функцією. При цьому така змінна стає глобальною не після опису, а після виклику цієї функції, наприклад:

```
function f()
{ var i=5; k=7; }

f()
```

У наведеному прикладі після **i = 5** стоїть крапка з комою (а не кома). Отже, **k = 7** – це окреме оголошення змінної, вже без оператора **var**. Тому змінну **k** "видно" зовні, і її значення (7) буде виведено оператором **alert (k)**. Щоб приховати змінну **k**, потрібно було після **i = 5** поставити кому.

Необхідно розглянути інший приклад, що показує, до яких несподіваних наслідків може призвести відсутність **var** при описі змінної:

```
function f(i)
{
k=7;
if(i==3) k=5;
else { f(3); alert(k); }
}
f(0)
```

При виклику **f(0)**, змінній присвоюється значення **k = 7**, далі виконання функції йде по гілці **else** – і оператор **alert(k)** видає **5** замість очікуваного **7**. Причина в тому, що виклик f (**3**) за "побічним ефектом" змінив значення **k**. Щоб такого не сталося, потрібно перед **k = 7** поставити **var**. Тоді змінна **k** буде локальною, і виклик **f(3)** не зможе її змінити, осільки при виклику функції створюються нові копії всіх її локальних змінних.

Під час написання великих програм подібні помилки важко відстежити, тому рекомендується всі змінні оголошувати з оператором **var**, особливо всередині функцій.

Масиви

Масиви розділяються на вбудовані (document.links [], document.images [] тощо – їх ще називають колекціями) і визначені користувачем (автором документа). Слід детально зупинитися на масивах, визначених користувачем. Для масивів призначено декілька методів: **join(), reverse(), sort()** та інші, а також властивість **length**, яка дозволяє отримати кількість елементів масиву.

Для визначення масиву користувача існує спеціальний конструктор **Array**. Якщо йому передається один аргумент, причому ціле невід'ємне число, то створюється незаповнений масив відповідної довжини. Якщо ж передається один аргумент, який не є числом, або більше одного аргументу, то створюється масив, заповнений цими елементами:

```
a = new Array(); // порожній масив (довжини 0)
b = new Array(10); // масив довжини 10
c = new Array(10,'Привіт'); // масив з двох елементів: числа та рядки
//Короткий спосіб створити масив з 4-х елементів
d = [5, 'Тест', 2.71828, 'Кількість e'];
```

Елементи масиву нумеруються від нуля. Тому в останньому прикладі значення **d [0]** одне **5**, а значення **d [1]** одне **'Тест'**. Отже, масив може складатися з різнорідних елементів. Масиви не можуть бути багато вимірюваними, проте можна завести масив, елементами якого також будуть масиви.

Метод join()

Метод join() дозволяє об'єднати елементи масиву в один рядок. Він є зворотним до розглянутого вище методу split(), який розрізає об'єкт типу String на шмати та складає з них масив. До речі, метод split() демонструє той факт, що масив можна отримати і без конструктора масиву.

Доцільно розглянути приклад перетворення локального URL на глобальний URL, де в якості адреси сервера виступатиме www.intuit.ru. Якщо у змінній localURL зберігається локальний URL певного файла:

```
localURL = "file:///C:/department/internet/js/2/2.html"
```

Якщо розірвати рядок у місцях входження комбінації символів "":/" за виконанням команди: b = localURL.split (':/'), отримується масив:

```
b[0] = "file";
b[1] = "//C";
b[2] = "department/internet/js/2/2.html";
```

Нульовий і перший елементи замінені на необхідні:

```
b[0] = "http:";
b[1] = "/www.intuit.ru";
```

Нарешті, склеється отриманий масив, вставляючи косу риску в місцях склеювання: globalURL = b.join ("/"). У результаті отримано необхідний глобальний URL – значення globalURL дорівнюватиме: http://www.intuit.ru/department/internet/js/2/2.html.

Метод reverse()

Метод reverse() застосовується для зміни порядку елементів масиву на протилежний. Якщо, масив впорядкований у порядку зростання:

```
a = new Array('мати', 'бачить', 'дочку');
```

Впорядкувати його у зворотному порядку можна, викликавши метод a.reverse(). Тоді новий масив **a** буде містити:

```
a[0]= 'дочка';
a[1]= 'бачить';
a[2]= 'мати';
```

Метод sort()

Метод sort() інтерпретує елементи масиву як рядкові літерали та сортує масив у алфавітному (лексикографічному) порядку. Слід зауважити, що метод sort() змінює масив. У попередньому прикладі, застосувавши a.sort(), отримано на виході:

```
a[0]= 'бачить';
a[1]= 'дочка';
a[2]= 'мати';
```

Однак це незручно, якщо потрібно відсортувати числа, оскільки за алфавітним порядком 40 йде раніше за 5. Для цих цілей у метода sort() є необов'язковий аргумент, який є ім’ям функції, згідно з якою потрібно відсортувати масив, тобто в цьому випадку виклик методу має вигляд: a.sort(myfunction). Ця функція повинна задовольняти певним вимогам:

вона повинна мати тільки два аргументи;

функція повинна повертати число;

якщо перший аргумент функції повинен вважатися меншим (більшим, рівним) за другий аргумент, то функція повинна повернути від'ємне (позитивне, нуль) значення.

Наприклад, якщо потрібно сортувати числа, то наступна функція описується:

```
function compar(a,b)
{
  if(a < b)  return -1;
  if(a > b)  return  1;
  if(a == b) return  0;
}
```

Тепер, якщо є масив b = new Array (10,6,300,25,18), то можна порівняти результати сортування без аргументу з функцією *compar* як аргумент:

```
document.write("Алфавітний порядок:<BR>");
document.write(b.sort());
document.write("<BR>Числовий порядок:<BR>");
document.write(b.sort(compar));
```

У результаті виконання цього коду отримано наступне:
Алфавітний порядок:
10,18,25,300,6
Числовий порядок:
6,10,18,25,300

Слід зауважити, що метод sort() інтерпретує елементи масиву як рядки (і здійснює лексикографічне сортування), але не перетворює їх на рядки. Якщо в масиві були числа, то вони залишаться числами. У цьому легко переконатися, якщо наприкінці останнього прикладу виконати команду **document.write (b [3] +1)**: результат буде 26 (тобто 25 +1), а не 251 (тобто "25" +1).

Оператори мови

У цьому розділі будуть розглянуті оператори JavaScript. Основна увага при цьому приділяється операторам декларування й управління потоком обчислень. Без них не може бути написана жодна JavaScript-програма.

Перелік основних операторів виглядає наступним чином:
{...}
if... else...
()?
while
for
break
continue
return

Оператор {...}

Фігурні дужки визначають складений оператор JavaScript-блок. Основне призначення блоку – визначення тіла циклу, тіла умовного оператора або функції.

Оператор if... else...

Умовний оператор застосовується для розгалуження програми за певною логічною умовою. Є два варіанти синтаксису:

```
if (логічний_вираз) оператор_1;
if (логічний_вираз) оператор_1; else оператор_2;
```

Логічний вираз – це вираз, який приймає значення **true** або **false**. У першому варіанті синтаксису: якщо логічний_вираз одне true, то виконується вказаний оператор. У другому варіанті синтаксису: якщо логічний_вираз одне true, то виконується оператор_1, якщо ж воно дорівнює false оператор_2. Приклад використання (про об'єкт **navigator** в лекції "Програмуємо властивості вікна браузера"):

```
if (navigator.javaEnabled())
  alert('Ваш браузер підтримує Java');
else
  alert('Ваш браузер підтримує Java');
```

Оператор()?

Цей оператор, званий умовним виразом, видає одне з двох значень залежно від виконання певної умови. Синтаксис його такий:

```
логічний_вираз)? значення_1 : значення_2
```

Якщо логічний_вираз одне **true**, то повертається значення_1, в іншому випадку – значення_2. Умовний вираз легко імітується оператором **if... else**, проте він дозволяє зробити більш компактним і легко сприйнятним код програми. Наприклад, наступні два фрагмента рівнозначні:

```
TheFinalMessage = (k>5)? 'Готово!' : 'Зачекайте...';

if(k>5)
  TheFinalMessage = 'Готово!';
else
  TheFinalMessage = 'Зачекайте...';
```

Оператор while

Оператор while задає цикл, який визначається у загальному випадку наступним чином:

while (умова_продовження_циклу) тіло циклу**;**

Тіло циклу може бути як простим, так і складним оператором. Складний оператор, як правило, вкладається у фігурні дужки. Рекомендується і простий оператор укладати в них, щоб програму можна було легко модифікувати. Умова_продовження_циклу є логічним виразом. Тіло виповнюється доти, доки правильною є логічна умова. Формально цикл while працює таким чином:

1. Перевірити умову_ продовження_циклу.
2. Якщо вона помилкова (false), цикл закінчений.
3. Якщо ж істинна (true), то продовжувати далі.
4. Виконати тіло_циклу;
5. Перейти до пункту 1.

Такий цикл використовується, коли заздалегідь невідома кількість ітерацій, наприклад в очікуванні деякої події. Приклад:

```
var s='';
while (s.length<6)
{
   s=prompt('Введіть рядок довжини не менше 6:','');
}
alert('Ваш рядок: ' + s + '. Дякую!');
```

Оператор for

Оператор **for** – це ще один оператор циклу. У загальному випадку він має вигляд:

```
for (ініціалізація_змінних_циклу;
   умова_продовження_циклу;
   модифікація_змінних_циклу)
      тіло_циклу;
```

Тіло циклу може бути як простим, так і складним оператором (складні необхідно укладати у фігурні дужки). Оператори ініціалізація_змінних_циклу і модифікація_змінних_циклу можуть складатися з декількох простих операторів, у цьому випадку прості оператори повинні бути розділені комою. Умова_ продовження_циклу є логічним виразом. Цикл *for* працює таким чином:

1. Виконується ініціалізація_змінних_циклу.
2. Перевіряється умова_ продовження_циклу.
3. Якщо вона помилкова (false), цикл закінчений.
4. Якщо ж істинна (true), то продовжувати далі.
5. Виконується тіло_циклу.
6. Виконується модифікація_змінних_циклу.
7. Слід перейти до пункту 2.

Типовий приклад використання цього оператора:

```
document.write('Куби чисел від 1 до 100:');

for (n=1; n<=100; n++)
document.write('<BR>'+n+'<sup>3</sup> = '+ Math.pow(n,3));
```

Тут **Math** – вбудований об'єкт, що дає численні математичні константи та функції, а Math.pow (n, m) обчислює ступеневу функцію nm. Результат роботи скрипта пропонується отримати самостійно.

Оператор break

Оператор break дозволяє достроково покинути тіло циклу. Із прикладу з кубами чисел роздруковуються тільки куби, що не перевищують 5 000.

```
document.write('Куби чисел, менші 5000:');
for (n=1; n<=100; n++)
{  s=Math.pow(n,3);
   if(s>5000) break;
   document.write('<BR>'+n+'<sup>3</sup> = '+s);}
```

Незважаючи на те, що змінну n змусили пробігати від 1 до 100, тобто з запасом, реально ж цикл виконується для значень n від 1 до... Ці значення пропонується отримати самостійно.

Оператор continue

Оператор continue дозволяє перейти до наступної ітерації циклу, пропустивши виконання всіх нижчерозташованих операторів в тілі циклу. Якщо потрібно вивести куби чисел від 1 до 100, що перевищують 10 000, то можна скласти такий цикл:

```
document.write('Куби чисел від 1 до 100, більше 10 000:');
for (n=1; n<=100; n++)
{
  s=Math.pow(n,3);
  if(s <= 10000) continue;

  document.write('<BR>'+n+'<sup>3</sup> = '+s);
}
```

Перевірте самостійно, куби яких чисел будуть виведені скриптом. Зрозуміло, для більшої гнучкості можна використовувати в циклах обидва оператори: **break** і **continue**.

Оператор return

Оператор return використовують для повернення значення з функції або обробника події. Приклад з функцією:

```
function sign(n)
{
  if (n>0) return 1;
  if (n<0) return -1;
  return 0;
}
alert( sign(-3) );
```

Зверніть увагу: оператор **return** не тільки вказує, яке значення має повернути функція, а й припиняє виконання подальших операторів в тілі функції.

Під час використання в обробниках подій оператор **return** дозволяє скасувати або не скасовувати дію за замовчуванням, яка створює браузер під час виникнення даної події. Скасувати його, однак, можна не для всіх подій, наприклад:

```
<FORM ACTION="newpage.html" METHOD=post>
  <INPUT TYPE=submit VALUE="Відіслати?"
        onClick="alert('Не відішлемо!'); return false;">
</FORM>
```

У цьому прикладі без оператора **return false** користувач побачив би вікно попередження "Не відішлемо!" і далі був би перенаправлений на сторінку **newpage.html**. Оператор же **return false** дозволяє скасувати відіслання форми, і користувач побачить лише вікно попередження.

Аналогічно, щоб скасувати дію за замовчуванням для подій **onClick**, **onKeyDown**, **onKeyPress**, **onMouseDown**, **onMouseUp**, **onSubmit**, **onReset**, потрібно використовувати **return false**. Для події **onMouseOver** з цією ж метою потрібно використовувати оператор **return true**. Для деяких же подій, наприклад **onMouseOut**, **onLoad**, **onUnload**, скасувати дію за замовчуванням неможливо.

Налагодження коду на JavaScript

Процес налагодження є невід'ємною частиною розроблювання програм, зокрема написаних мовою JavaScript. Тому застосування ефективних технік налагодження коду підвищує швидкість розроблювання та якість програмного коду.

Дебагери (відладники)

На даний момент інструменти налагодження існують для всіх основних веб-браузерів:

для Firefox існує добре відоме розширення Firebug;

IE8 виходить з вбудованим Developer Tools (засоби розробника);

Opera (версія 9.5 і вище) підтримує дебагер Opera Dragonfly;

в останніх версіях Safari дебагер вбудований в WebInspector;

у Google Chrome є консоль JavaScript.

На сьогодні Firebug і Dragonfly є найбільш стабільними варіантами вибору.

Процес налагодження

Алгоритм (приблизний) пошуку неполадок в коді складається з наступних кроків:

1. Пошук відповідного коду, панелі перегляду коду дебагера.
2. Установка точки зупинки (breakpoints) у місці потенційного виникнення помилки.
3. Перезапуск скрипту, якщо він вбудований, оновлення сторінки в браузері.
4. Очікування, поки дебагер НЕ призупинить виконання і не дасть можливість переглянути код покроково.
5. Аналіз значень змінних. Наприклад, пошук тих змінних, які не визначені, але повинні містити значення, або повертають "false", тоді очікується "true".
6. За потребою, використання командного рядка для виконання коду або зміни значень змінних для тестування.
7. Знаходження проблеми за виділенням тої частини коду або введення, які викликають помилку.

Вимоги дебагера

Більшість дебагерів вимагає добре відформатованого коду. Скрипти, написані в один рядок, ускладнюють процес пошуку помилок у порядкової дебагера. Заплутаний код складно налагоджувати, особливо якщо він "запакований" і вимагає розпакування через eval(). Безліч бібліотек JavaScript дозволяє вибирати між упакованим/заплутаним і добре відформатованим варіантами, даючи можливість використовувати відформатований код для налагодження.

Демонстрація налаштування

Слід почати з невеликого, повного багів прикладу для того, щоб навчитися знаходити та послідовно виправляти кожну неполадку. Прикладом є екран входу до веб-додатка (рис. 4.10).

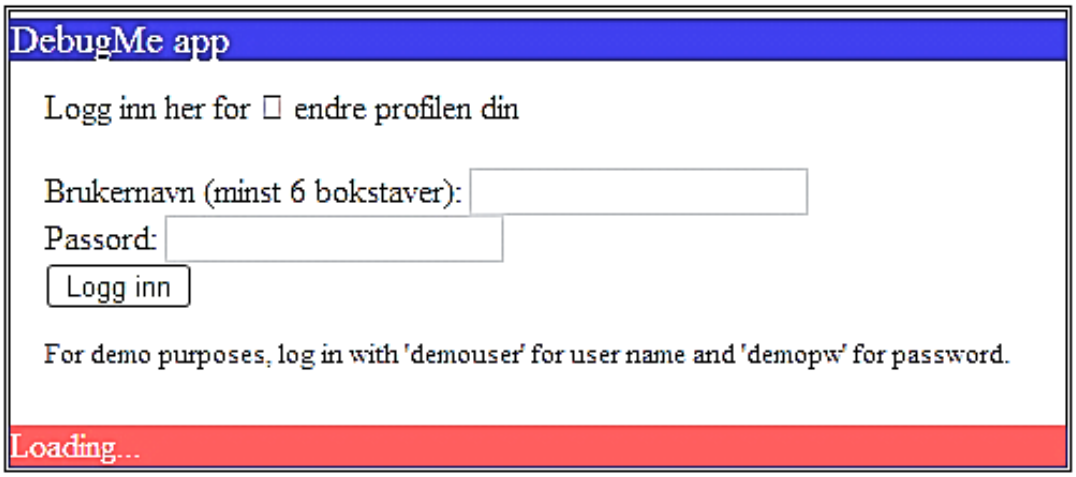


Рис. 4.10. **Екран входу в дебагер веб-додатка**

Уявіть, що ви працюєте з новим веб-додатком і ваші тестери просять вас розглянути такий список багів:

1. Повідомлення "Loading..." в статус барі не зникає після повного завантаження додатка.

2. Мова за замовчуванням – норвезька, навіть в англійських версіях Firefox і IE.

3. Десь виникає глобальна змінна prop.

Запуск дебагера

У Firefox необхідно упевнитися, що встановлено розширення Firebug. Для запуску потрібно вибрати "Інструменти > Firebug > Відкрити Firebug".

В Opera 9.5 і вище потрібно вибрати "Інструменти> Додатково> Засоби розробки (Opera Dragonfly)".

У IE8, потрібно вибрати "Сервіс> Засоби розробника".

У Safari або WebKit спочатку потрібно вибрати меню налагодження, а потім – "Налагодження> Показати Web Inspector".

У Google Chrome потрібно вибрати "Управління поточною сторінкою> Розробникам> Консоль JavaScript".

Баг перший: повідомлення "Loading..." в статус бар

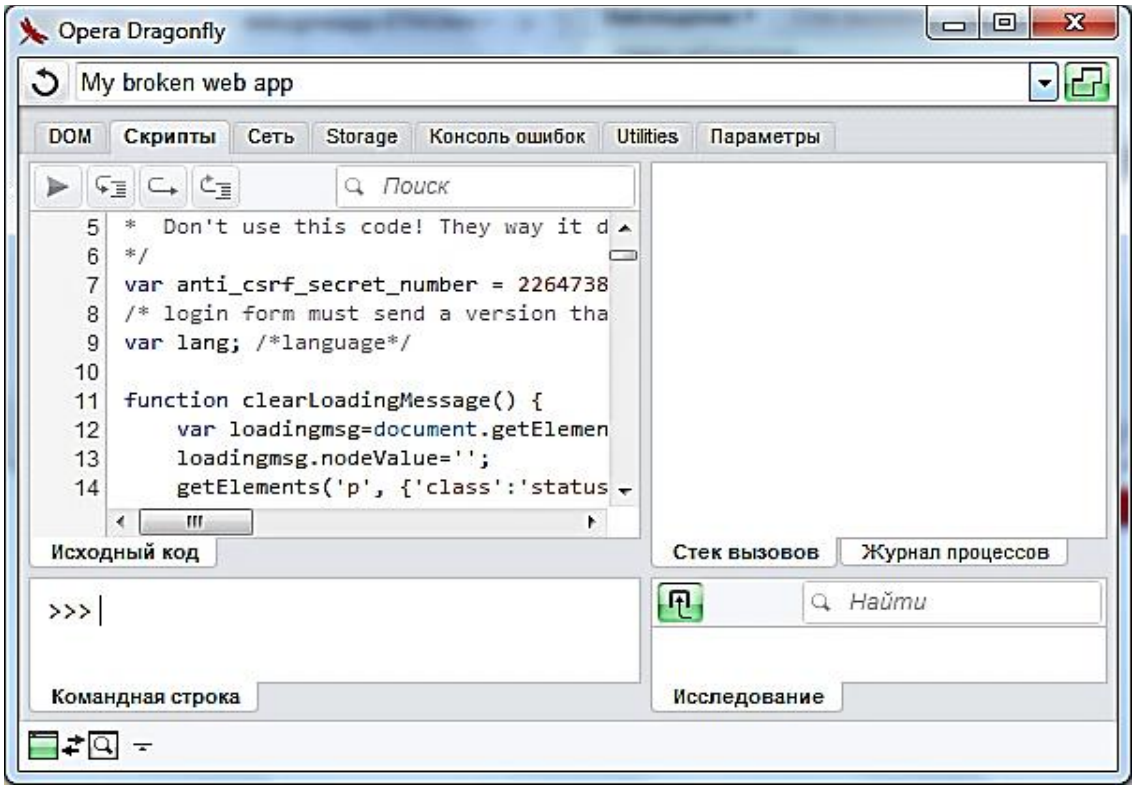
Первісний вигляд додатків налагодження в Dragonfly і Firebug наведено на рис. 4.11.

Якщо подивитися на вихідний код до дебагера, можна помітити, що там є функція clearLoadingMessage(), оголошена на початку коду. Це місце може бути слушним для точки зупинки (breakpoint).

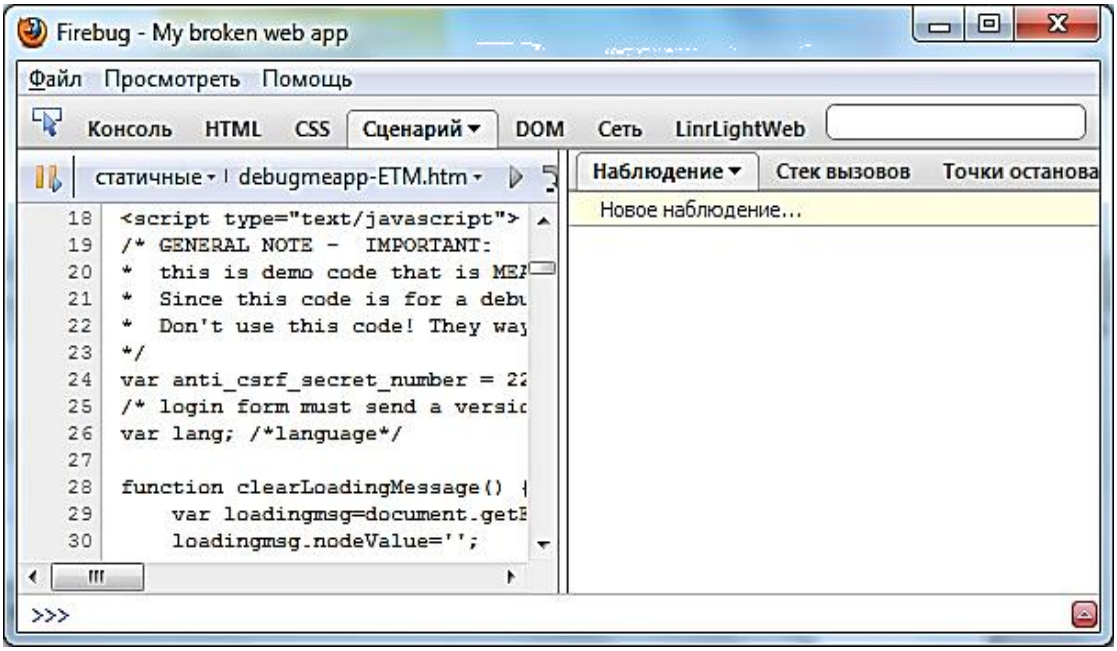
Для встановлення точки зупинки потрібно:

1. Клікнути на номері рядка в лівому полі для встановлення *breakpoint* в першому рядку всередині функції clearLoadingMessage();

2. Оновити сторінку.



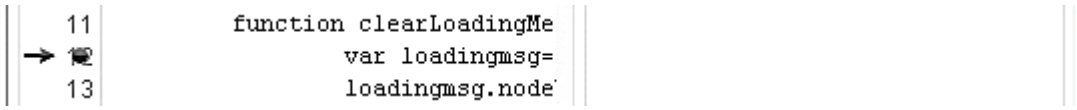
А. **Dragonfly**



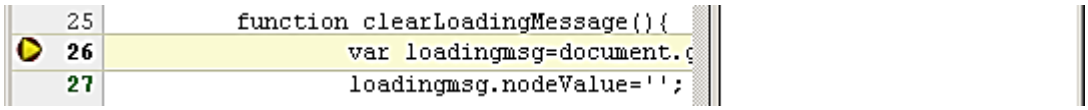
Б. **Firebug**

Рис. 4.11. **Початковий вигляд додатка JavaScript у відладнику**

Зауваження: точка зупинки повинна бути встановлена на рядку коду, який буде запускати функцію. Рядок містить "function clearLoadingMessage(). {..." є всього лише оголошенням функції. Установка breakpoint в цьому місці ніколи не викличе зупинки дебагера. Замість цього потрібно встановити точку зупинки на перший рядок всередині функції. Після оновлення сторінки, запуск скрипту буде зупинений на breakpoint, як на рис 4.12.



А. **Dragonfly**



Б. **Firebug**

Рис. 4.12. **Робота з точками зупинки**

Слід розглянути функцію покроково. Вона змінила два DOM елементу, і в рядку 28 згадується слово statusbar, яке виглядає як знак. Типу getElements ('p', {'class': 'statusbar'}). [0] знайде елемент statusbar в DOM. Чи є спосіб швидко перевірити це припущення?

Потрібно вставити відповідний блок в командний рядок для того, щоб перевірити припущення. На рис. 4.13 показано три знімка екранів (Dragonfly, Firebug і IE8) після читання innerHTML або outerHTML елемента, що повертаються командою, про яку сказано вище.

Для тестування цього припущення потрібно:

відкрити командний рядок;

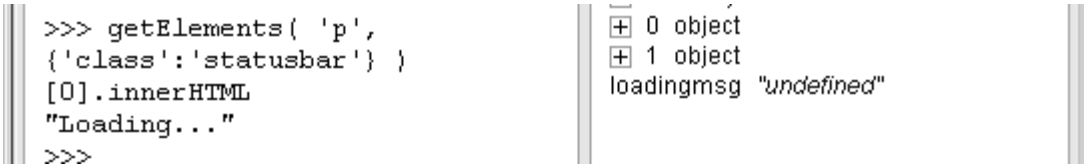
у Firebug, переключити вкладку "Console";

у Dragonfly під панеллю вихідного коду JavaScript;

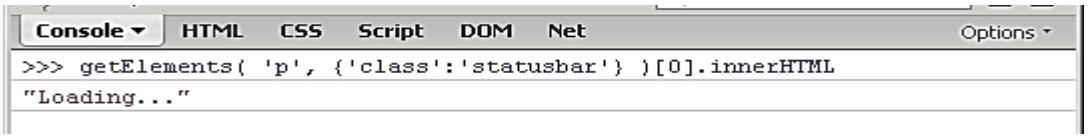
у IE8 Developer Tools, праворуч вкладка "Console".

вставити getElements ('p', {'class': 'statusbar'}) [0]. InnerHTML у командний рядок;

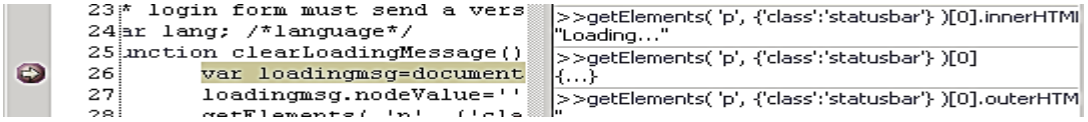
натиснути "Вхід".



А. **Dragonfly**



Б. **Firebug**



В. **IE8**

Рис. 4.13. **Налагодження: пошук елемента в DOM**

Командний рядок є зручним інструментом, який дозволяє швидко тестувати невеликі шмати коду. Вбудована консоль Firebug дуже зручна – якщо вивід команди є об'єктом, перегляд стає зрозумілим. Наприклад, якщо це об'єкт DOM, отримується відповідна розмітка.

Можна використовувати командний рядок для глибокого розгляду проблеми. Рядок JavaScript створює наступне:

1) отримує посилання на елемент statusbar. У DOM inspector view відповідною розміткою буде <p class="statusbar">;

2) показує, що це firstChild, іншими словами, перший елемент всередині елемента параграф;

3) установлює властивість innerText.

Якщо ускладнити операцію з командою рядка (використовуючи кнопку "Стрілка вгору" для повернення до попередніх команд, які набирались у полі командного рядка), можна, наприклад, дізнатися, яким є поточне значення властивості innerText елемента до того, як воно встановлюється. Щоб перевірити це, можна набрати наступну команду:

```
getElements('p', {'class':'statusbar'})[0].firstChild.innerText
```

На диво, виведеться... ніщо. Отже, вираз getElements ('p', {'class': 'statusbar'}) [0]. FirstChild відноситься до чогось в DOM, що не містить тексту або не має властивості innerText.

Постає, наступне питання: що саме є першою нодою в елементі параграфа? Це питання вирішується у командному рядку (результат наведено на рис. 4.14)

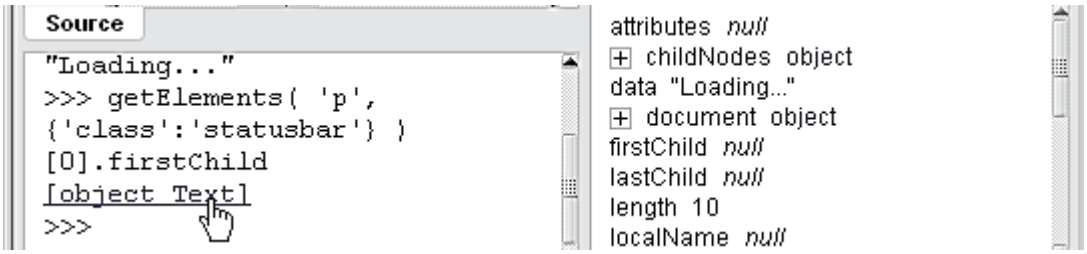


Рис. 4.14. **Командний рядок дебагера Dragonfly, виводить [object Text]**

Висновок [object Text] від Dragonfly показує, що це текстовий вузол DOM. Firebug показує вміст текстового вузла як посилання на оглядач DOM. Після знаходження проблеми, яка пояснює перший баг: текстові вузли не мають властивості innerText - вони є тільки у вузлів елементів. Отже, установка p.firstChild.innerText є бездіяльна. Цей баг може бути усунутий заміною innerText на nodeValue, що є властивістю, визначеною стандартом W3C DOM для вузлів тексту.

Після чого виникає можливість переглянути цей приклад:

1) натиснути [F5] або кнопку запуску для завершення скрипта, коли проблема вже виявлена;

2) не забути прибрати всі старі точки зупинки повторним натисканням на номери рядків.

Баг другий: виявлення проблемної мови

Рядок var lang;/* language */вгорі JavaScript – код, який встановлює цю змінну, напевне, пов'язаний, якщо виконання відбувається невірно. Можна швидко знаходити певні речі в коді, використовуючи ручні функції пошуку, подані обома дебагерами. У Dragonfly це знаходиться одразу над переглядом коду; в Firebug – у правому верхньому кутку для користувача інтерфейсу (рис. 4.15).

Для знаходження місць, які мають відношення до локалізації:

1) набрати "lang =" в рядку пошуку;

2) встановити точку зупинки у рядку, де визначається значення для змінної lang;

3) оновити сторінку.

WebInspector Safari також має багаті можливості пошуку. WebInspector дозволяє шукати по всьому коду, включаючи розмітку, CSS і JavaScript. Результати показуються в окремій панелі, з якої можна перейти до вихідного коду за допомогою подвійного кліка, як показано на скріншоті.

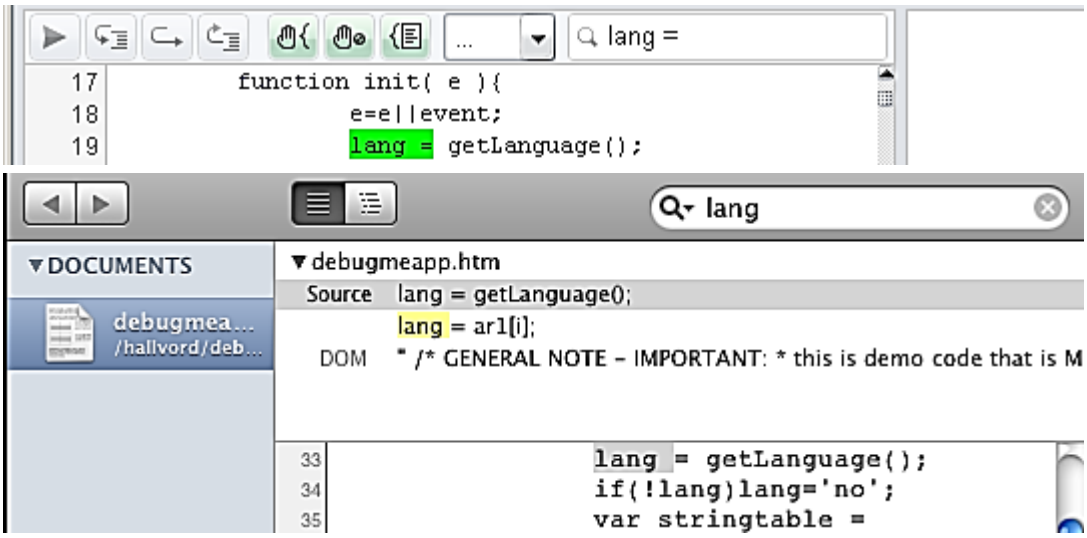


Рис. 4.15. **Пошук за допомогою дебагера *Dragonfly* й *WebInspector***

Для перевірки роботи цієї функції:

1) слід використовувати кнопку "step into" для введення функції getLanguage;

2) потрібно натиснути повторно кнопку "step into" для проходу по коду з кроком в один рядок за один раз;

3) необхідно спостерігати за локальними змінними, щоб бачити, як вони змінюються в процесі виконання функції.

Дійшовши до функції *getLanguage*, можна побачити, що вона намагається зчитувати мову з рядка *user-agent*. Автор цього коду помітив, що деяка інформація про мову включена в рядок *user-agent* в деяких браузерах, і намагається парсити *navigator.userAgent,* щоб отримати інформацію:

```
var str1 = navigator.userAgent.match( /\((.*)\)/ )[1];
var ar1 = str1.split(/\s*;\s*/), lang;
for (var i = 0; i &lt; ar1.length; i++){
    if (ar1[i].match(/^(.{2})$/)){
        lang = ar1[i];
    }
}
```

Проходячи по цьому коду за допомогою step в дебагері, можна використовувати огляд локальних змінних. На рис. 4.16 показано *Firebug* і *IE8 Developer Tools* з розгорнутим масивом *ar1* для демонстрації елементів у ньому.

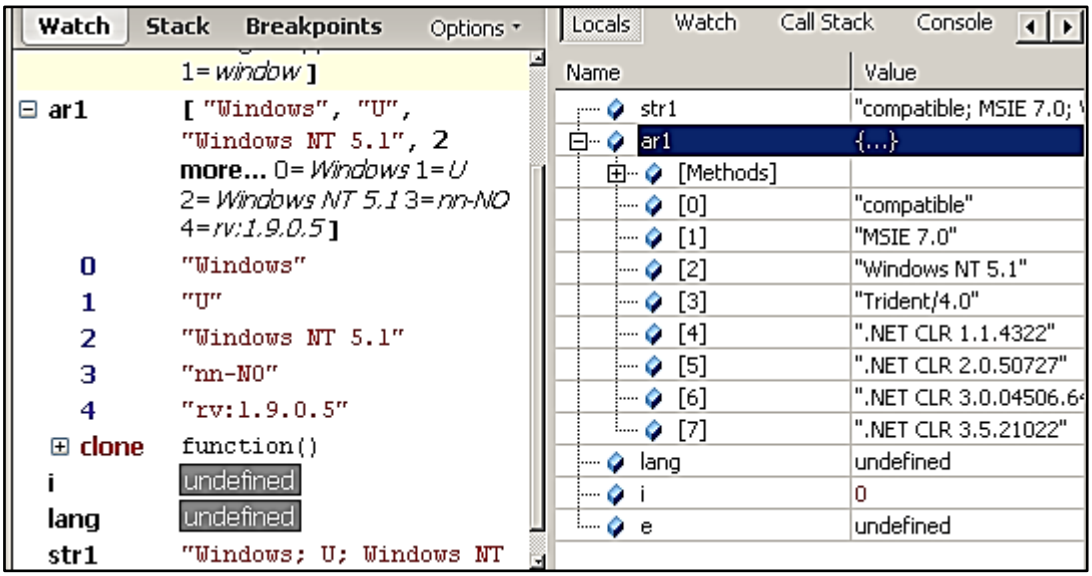


Рис. 4.16. **Панель локальних змінних *Firebug* і *IE8* під час роботи функції *getLanguage***

Затвердження ar1 [i]. Match (/ ^ (. {2}) $ /) просто шукає рядок розміром у два символи, очікуючи зустріти два символи коду мови, як en або no. Як можна побачити на скріншоті, інформація про мову Firefox у рядку user-agent дорівнює nn-NO. В IE відсутня інформація, що відноситься до мови, в цій частині рядка userAgent.

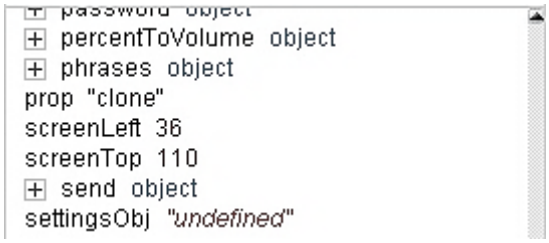
Другий баг був знайдений: виявлення мови шукає два символи, що позначають код мови, але Firefox містить п'ять букв рядка locale, а IE не має її взагалі. Цей код "виявлення мови", вірогідно, повинен бути зібраний разом і замінений на виконуване на боці сервера, який

використовує header HTTP Accept-Language або, якщо можливо, читати navigator.language або navigator.userLanguage в IE. Приклад того, як ця функція може виглядати:

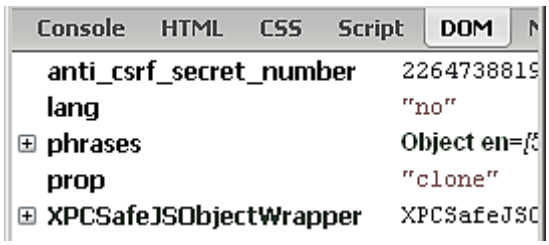
```
function getLanguage() {
    var lang;
    if (navigator.language) {lang = navigator.language;}
    else if (navigator.userLanguage) {lang =
navigator.userLanguage;}

    if (lang && lang.length > 2) {lang = lang.substring(0, 2);}
    return lang;
}
```

Баг третій: незрозуміле виникнення змінної prop

На панелі локальних змінних (рис. 4.17) можна бачити список змінних.



А. **Dragonfly**



Б. **Firebug**

Рис. 4.17. **Панель локальних змінних в Firebug і Dragonfly показує глобальну змінну prop**

У правильно написаних додатках кількість глобальних змінних зводиться до мінімуму, позаяк вони можуть призводити до скрутних ситуацій, коли різні секції програми намагаються використовувати однакові імена для змінних. Якщо інша команда розробників додасть нову частину до того ж додатка і також назве її *prop*, то дві різні частини програми намагатимуться використовувати одну й ту ж змінну для різних цілей. Така практика призводить до конфліктів і появи багів. Отже, потрібно знайти, де ця змінна оголошена, і спробувати зробити її локальною. Можна почати шукати її у той же спосіб, яким вирішено другий баг. Проте є більш доцільний спосіб.

Дебагери в багатьох інших мовах програмування мають концепцію "watch", яка здатна сповіщати дебагер про зміни будь-якої змінної. Dragonfly та Firebug не підтримують "watch", але можна легко домогтися аналогічного ефекту, додавши наступний рядок коду дебагера, вгорі коду скрипта, який треба відстежити:

```
__defineSetter__('prop', function() { debugger; });
```

Для додавання "watch"-функціональності до налагодження скрипту:

1) додати вищевказаний код вгорі першого скрипта;
2) оновити сторінку;
3) відстежити, як він реагує, при знаходженні проблеми.

При використанні getters і setters можна емулювати функціональність "watch" і допомогти в "розумній" установці breakpoint'ів.

У IE8 Developer Tools є панель "watch", але вона не здіснює переривання, коли змінна була змінена. Підтримка getters і setters, яка є у IE8, не дає можливості емулювати функціональність тим способом, яким можна зробити це в Firefox, Opera і Safari.

Якщо відновити додаток, відбувається переривання після оголошення глобальної змінної prop. Тоді, зупинка буде здійснена на коді, який викликає налагодження, тому що саме там йде вираз дебагера. Одним натисканням на кнопку "step out" відбувається перехід від функції setter до того місця, де була оголошена змінна. Цей код був знайдений усередині функції getElements:

```
for (prop in attributes) {
  if (el.getAttribute(prop) != attributes[prop])
    includeThisElement = false;
```

Тепер зупинка сталась в рядку, який починається з "for (prop". Тут можна побачити, що змінна prop використовується без попереднього оголошення як локальна змінна з ключовим словом var усередині функції. Її проста зміна на "for (var prop", вирішить третій баг.

4.4. Завдання до лабораторної роботи

1. Для реалізації завдань додати на сайт, створений в рамках лабораторної роботи № 3, сторінку з назвою "JavaScript.html".

2. Створити форму з полем уведення та двома кнопками (типу submit і reset). Натисканням на кнопку типу submit у формі заповнити текстове поле інформацією (а) відповідно до варіанту.

3. Створити файл "lr4.js", в якому реалізувати завдання (б) відповідно до варіанту, і підключити цей файл до JavaScript.html.

4. Додати елемент (в), відповідно до варіанту і в обробнику події (г) цього елемента реалізувати завдання (д). У обробнику повинна викликатися функція з ім’ям "js_event()", реалізація якої повинна бути в контейнері <SCRIPT>.

5. Додати гіперпосилання, при натисканні на яке у вікні вивести інформацію (е) відповідно до варіанту.

Варіанти завдання

Варіант 1

А. Кількість днів (обчислити в скрипті), що минули від дня вашого народження.

Б. Написати скрипт, який визначає кількість днів до найближчої неділі, і вивести це число під час завантаження сторінки у вікні повідомлення.

В. Кнопка.

Д. Втрата клавіатурного фокусу.

Е. Висновок в рядку стану назви поточного місяця російською (українською) мовою.

Ж. Список (одним рядком) днів тижня без букви, з якої починається кількість натиснень на дане гіперпосилання в поточному сеансі.

Варіант 2

А. Кількість тижнів (обчислити в скрипті), які пройшли від дня вашого народження.

Б. Написати скрипт, який визначає, скільки днів минуло з Нового року, і вивести це число при завантаженні сторінки у вікні повідомлення.

В. Поле введення.

Д. Отримання клавіатурного фокусу.

Е. Висновок в рядку стану за кількістю годин і хвилин, що пройшли з початку доби.

Ж. Список (одним рядком) місяців, в яких менше 31-го дня. Назви місяців мають бути відокремлені один від одного символом "/".

Варіант 3

А. Кількість годин і хвилин, що пройшли з початку поточного місяця.

Б. Написати скрипт, який визначає кількість годин (за ґрінвічським часом) до настання Нового року, і вивести це число при завантаженні сторінки у вікні повідомлення.

В. Зображення.

Д. Наведення курсору миші на елемент.

Е. Висновок в рядку стану поточних координат курсора.

Ж. URL сайта дистанційної освіти ХНЕУ без символів "/". Застосувати методи об'єкта Array**.**

Варіант 4

А. Повна інформація щодо поточної дати та часу. Наприклад, "14 травня 2002 року, вівторок, 2:53:44 pm".

Б. Написати скрипт, який визначає, скільки тижнів залишилося до 1 вересня поточного року.

В. Комбінований список.

Д. Зміна елемента списку.

Е. Висновок в рядку стану значення поточного елемента списку.

Ж. Вміст поля введення форми без пробілів.

Варіант 5

А. Кількість тижнів, що минули з останнього 1 вересня.

Б. Написати скрипт, який визначає, чи є поточний рік роком проведення літньої Олімпіади (високосним), і вивести це число під час завантаження сторінки у вікні повідомлення.

В. Список.

Д. Натискання клавіші (тільки.'←' або .'←').

Е. Висновок в рядку стану значення поточного елемента списку.

Ж. Список (одним рядком) назв місяців без букви, на яку закінчується кількість натиснень на дане гіперпосилання в поточному сеансі.

Варіант 6

А. Кількість годин (обчислити в скрипті), що залишилися до початку літа.

Б. Написати скрипт, який визначає, скільки діб залишилось до днів весняного й осіннього рівнодення (22 березня та 22 вересня), і вивести це число при завантаженні сторінки у вікні повідомлення.

В. Поле вибору (checkbox).

Д. Вибір.

Е. Висновок в рядку стану щодо кількості спрацьовувань обробника даної події в поточну хвилину.

Ж. Список перших десяти простих чисел, об'єднаних символом "+".

Варіант 7

А. Кількість годин і хвилин, що пройшли з початку поточного місяця.

Б. Написати скрипт, який визначає:

півріччя (перше або друге);

квартал (перший, другий, третій чи четвертий);

сезон (зима, весна, літо або осінь);

сторіччя;

тисячоліття;

і вивести цю інформацію під час завантаження сторінки у вікні повідомлення.

В. Тема першого рівня.

Д. Подвійний клік.

Е. Висновок в рядку стану поточної секунди.

Ж. Поточна хвилина.

Варіант 8

А. Скільки днів залишилося до найближчої п'ятниці, яка випадає на 13-те число.

Б. Написати скрипт, який визначає, скільки днів залишилось до найближчого 23 серпня.

В. Важливий текст.

Д. Наведення курсору миші.

Е. Висновок в рядку стану поточного часу.

Ж. Випадкове число від 1 900 до 2 010.

Варіант 9

А. Кількість днів, що залишились до літніх канікул (уточніть дату початку канікул в деканаті).

Б. Написати скрипт, який визначає кількість годин, що залишились до кінця поточного місяця, і вивести це число при завантаженні сторінки у вікні повідомлення.

В. Перемикач (radiobutton).

Д. Відпускання кнопки миші (MouseUp).

Е. Висновок у рядку стану випадкового числа від 1 до кількості днів у поточному місяці.

Ж. День тижня останнього дня поточного місяця.

Варіант 10

А. Кількість хвилин, що минули з початку пари.

Б. Написати скрипт, який визначає, скільки днів минуло з останнього 14 лютого.

В. Вікно браузера.

Д. Зміна розміру.

Е. Висновок в рядку стану щодо кількості спрацьовувань обробника даної події в поточному сеансі.

Ж. Значення атрибута 'Title' даного тегу '<a>'.

4.5. Контрольні запитання

1. Назвіть способи розміщення коду JavaScript на HTML-сторінці.
2. Як додати коментарі до коду JavaScript?
3. Чому JavaScript є слабоуніверсальна мова?
4. Як дізнатися тип змінної в JavaScript?
5. Що таке літерали?
6. Для чого використовують оператор ***var***?
7. Назвіть особливості використання оператора ***var*** усередині функцій.
8. Які види масивів існують в JavaScript?
9. Назвіть методи об'єкта "масив" і призначення цих методів.
10. Які вам відомі оператори в JavaScript?
11. Назвіть послідовність дій при налагодженні коду на JavaScript і прокоментуйте кожну.

Лабораторна робота № 5
Розроблення динамічних веб-сторінок за допомогою мови JavaScript та DOM API

5.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення об'єктної моделі документа та засобів роботи з нею в мові сценаріїв JavaScript при розробленні динамічних веб-сайтів.

5.2. Рекомендації щодо підготовки до виконання лабораторної роботи

У процесі підготовки до лабораторної роботи необхідно вивчити принципи об'єктної моделі документа та засоби роботи з нею в мові сценаріїв JavaScript.

5.3. Загальні тези лабораторної роботи

Для реалізації інтерактивних, динамічних веб-сторінок застосовується так званий Dynamic HTML, або DHTML [1 – 4].

Dynamic HTML – це спосіб створення інтерактивного веб-сайту, який використовує поєднання статичної мови розмітки HTML, вбудованої (і виконуваної на боці клієнта) зі скриптовою мовою JavaScript, CSS (каскадних таблиць стилів) і DOM (об'єктною моделлю документа) [3].

Такі web-технології, як HTML, CSS та основи JavaScript, вже були розглянуті в рамках курсу "Веб-програмування". Але перш ніж перейти до розгляду складової DHTML – DOM, слід розглянути об'єкти в мові JavaScript.

Об'єкти в JavaScript

Об'єкт **–** це головний тип даних JavaScript. Будь-який інший тип даних має об'єктову "обгортку" (wrapper). Це означає, що перш ніж отримати доступ до значення змінної того чи іншого типу, відбувається конвертація змінної в об'єкт, і тільки після цього виконуються дії над значенням. Тип даних Object сам визначає об'єкти [12; 15].

Об'єкт в JavaScript є звичайним асоціативним масивом ("хеш"). Він зберігає будь-які відповідності "ключ => значення" і має кілька стандартних методів. У сценарії JavaScript можуть використовуватися об'єкти декількох видів:

клієнтські об'єкти входять до моделі DOM, тобто відповідають тому, що міститься або відбувається на Web-сторінці у вікні браузера. Вони створюються браузером під час розбору (парсингу) HTML-сторінки. Приклади: window, document, location, navigator тощо;

серверні об'єкти відповідають за взаємодію клієнт-сервер. Приклади: Server, Project, Client, File тощо. Серверні об'єкти в цьому курсі не розглядаються;

вбудовані об'єкти – це різноманітні типи даних, властивостей, методів, притаманних самій мові JavaScript незалежно від вмісту HTML-сторінки. Приклади: вбудовані класи об'єктів Array, String, Date, Number, Function, Boolean, а також вбудований об'єкт Math;

користувальницькі об'єкти створюються програмістом у процесі написання сценарію з використанням конструкторів типу об'єктів (класу). Наприклад, можна створити свої класи Cat і Dog. Створення та використання таких об'єктів буде розглянуто далі.

Оператори роботи з об'єктами for... in...

Оператор **for** (змінна **in** об'єкта) дозволяє "пробігтися" по властивостям об'єкта, наприклад (об'єкт **document** розглянуто нижче):

```
for(v in document)
  document.write("document."+v+" = <B>"+
                                  document[v]+"</B><BR>");
```

Результатом роботи цього скрипта буде довгий список властивостей об'єкта document, наведено лише його початок (отримати його повністю пропонується самостійно):

```
alinkColor = #0000ff
bgColor = #ffffff
mimeType = HTML Document
defaultCharset = windows-1251
lastModified = 07/16/2002 21:22:53
onclick = null
links = [object]
```

*Примітка. Я*кщо запустити цей скрипт у різних браузерах, то можна побачити, що набір властивостей об'єкта **document** різний в різних

браузерах. Аналогічна ситуація з багатьма об'єктами моделі DOM, про яку піде мова нижче. Саме тому доводиться постійно дбати про так звані крос-браузерності – сумісності під час програмування динамічних HTML-документів.

with

Оператор **with** задає об'єкт за замовчуванням для блоку операторів, визначених у його тілі. Синтаксис його такий:

with (об'єкт) оператор;

Властивості та методи, притаманні тілу оператора, повинні бути записаними повністю, інакше вони вважатимуться властивостями та методами об'єкта, зазначеного в операторі **with**. Наприклад, якщо в документі є форма з ім’ям **anketa**, в якій є поля вводу з іменами age і speciality, то можна скористатися оператором **with** для скорочення запису:

```
with (document.anketa)
{
  age.value=35;
  speciality.value='програміст';
  window.alert(length);
  submit();
}
```

Тут **age.value** є скороченим зверненням до document.anketa.age.value; **length** є стислим записом властивості document.anketa.length (що означає кількість полів у формі); **submit**() є стислим записом методу document.anketa.submit() (що відсилає введені у форму дані на сервер), тоді як метод **window.alert**() записаний повністю і не належить до об'єкта document.anketa.

Оператором with корисно користуватися під час роботи з об'єктом Math, використовуваним для доступу до математичних функцій і констант. Наприклад, всередині тіла оператора with (Math) можна сміливо писати: sin (f) * cos (h + PI/2). Без оператора with довелося б вказувати Math три рази: Math.sin (f) * Math.cos (h + Math.PI/2).

Клієнтські об'єкти

Для створення механізму управління сторінками на клієнтському боці використовується об'єктна модель документа (DOM – Document Object Model). Сутність моделі в тому, що кожному HTML-контейнеру відповідає об'єкт, який характеризується трійкою: властивості; методи; події.

Об'єктну модель можна уявити як спосіб зв'язку між сторінками та браузером. Об'єктна модель документа – це подання об'єктів, їх методів, властивостей і подій, які присутні і відбуваються в програмному забезпеченні браузера, у вигляді зручного для роботи з ними з коду HTML і вихідного тексту сценарію на сторінці. З її допомогою можна ставити будь-які вимоги до браузера та передавати їх на клієнські сторінки. Браузер виконає команди і, відповідно, змінить сторінку на екрані.

Об'єкти з однаковим набором властивостей, методів і подій об'єднуються в класи однотипних об'єктів. *Класи* – це описи можливих об'єктів. Самі об'єкти з'являються тільки після завантаження документа браузером або як результат роботи програми. Про це треба завжди пам'ятати, щоб не звертатися до неіснуючого об'єкта.

Ієрархія класів DOM

Об'єктно-орієнтована мова програмування передбачає наявність ієрархії класів об'єктів. У JavaScript така ієрархія починається з класу об'єктів **Window**, тобто кожен об'єкт приписаний до того чи іншого вікна. Для звернення до будь-якого об'єкта або його властивості вказують повне або часткове ім’я цього об'єкта або його властивості, починаючи з імені об'єкта, старшого за ієрархією, до якої належить даний об'єкт (рис. 5.1).

Однак JavaScript не є класичною об'єктною мовою (її ще називають полегшеною об'єктною мовою). У неї відсутні успадкування та поліморфізм. Є лише тези "об'єкт А містить об'єкт В" (проілюстровано на рис. 5.1). Вона не є ієрархією класів у буквальному сенсі. Дійсно, розташування класу Window в цій ієрархії, вище від класу History, не означає, що будь-який об'єкт типу History є об'єктом типу Window та має всі його властивості та методи, як це розумілося б у стандартних об'єктно-орієнтованих мовах. У JavaScript це розташування означає, що об'єкт history є властивістю об'єкта window, а отже, щоб отримати до нього доступ, потрібно скористатися "точковою нотацією": window.history.

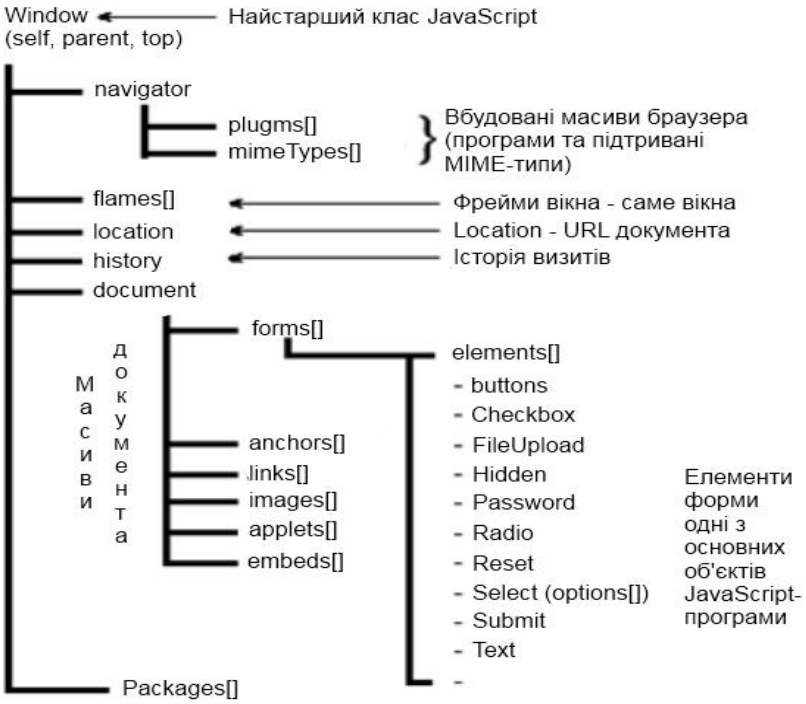


Рис. 5.1. **Ієрархія об'єктів DOM (фрагмент)**

У об'єктів DOM певні властивості обов'язково присутні, тоді як наявність інших залежить від веб-сторінки. Наприклад, об'єкт window завжди має в якості своїх властивостей об'єкти location і history, тобто це обов'язкові властивості. Якщо HTML-сторінка містить контейнер <BODY>, то в об'єкті window буде присутній як властивість об'єкт document. Якщо HTML-сторінка містить контейнер <FRAMESET> з вкладеними в нього контейнерами <FRAME>, то в об'єкті window будуть присутні в якості властивостей імена фреймів, наприклад window.f1. Останні самі є об'єктами класу Window, і для них, в свою чергу, справедливо все вищесказане.

Примітка. Кожен браузер, будь то Internet Explorer, Mozilla Firefox або Opera, має свою об'єктну модель. Об'єктні моделі різних браузерів (і навіть різні версії одного) відрізняються між собою, але мають принципово однакову структуру. Тому немає сенсу зупинятися на кожній з них окремо. Доцільно розглянути загальний підхід стосовно до всіх браузерів, загострюючи увагу на відмінностях між ними.

Колекції

Колекція – це структура даних JavaScript, подібна до масиву. Відмінність колекції від масивів полягає в тому, що масиви програміст створює сам в коді програми і заповнює їх даними. Колекції створюються браузером і "заселюються" об'єктами, пов'язаними з елементами веб-сторінки. Колекцію можна розглядати як більш зручний спосіб доступу до об'єктів веб-сторінки.

Наприклад, якщо на сторінці є форми з іменами f, g5 і h32, то в об'єкта document є відповідні властивості-об'єкти document.f, document.g5 тощо. Окрім цього, в об'єкта document є властивість forms, яка є колекцією (масивом) усіх форм. Отже, до тих самих об'єктів форм можна звернутися як document.forms[0], document.forms[1] тощо. Це зручно, при виконанні дій з усіма об'єктами форм на даній сторінці. При визначенні властивостей того чи іншого об'єкта колекції будуть писатись з дужками: forms[], images[], frames[], щоб підкреслити, що це не звичайні властивості, а колекції.

Нумеруються елементи колекції, починаючи з нуля, в порядку їх появи в початковому HTML-файлі. Доступ до елементів колекцій здійснюється або за індексом (в круглих або квадратних дужках), або за ім’ям (теж в круглих або квадратних дужках чи через точку), наприклад:

```
window.document.forms[4]     // 5-та форма на сторінці
window.document.forms(4)     // рівнозначно попередньому
window.document.forms['mf']  // форма з ім’ям 'mf'
window.document.forms('mf')  // рівнозначно попередньому
window.document.forms.mf     // рівнозначно попередньому
window.document.mf           // рівнозначно попередньому
```

Способи в 3 – 4-му рядках зручні, коли ім’я елемента колекції зберігається як значення змінної. Наприклад, якщо задати var w = "mf", то можна звернутися до форми з ім’ям "mf" як window.document.forms [w]. Саме таким чином створювався перелік усіх властивостей об’єкта document. у розділі оператор for... in.

Як і у звичайних масивів, у колекцій є властивість length, яка дозволяє дізнатись про кількість елементів в колекції. Наприклад, document.images.length. Основні колекції в об'єктній моделі документа наведено в табл. 5.1.

Таблиця 5.1

Колекції в об'єктній моделі документа

Колекція	Опис
window.frames[]	Усі фрейми, тобто об'єкти, що відповідають контейнерам <FRAME>
document.all[]	Усі об'єкти, що відповідають контейнерам всередині контейнера <BODY>
document.anchors[]	Усі якорі, тобто об'єкти, що відповідають контейнерам <A>
document.applets[]	Усі аплети, тобто об'єкти, що відповідають контейнерам <APPLET>
document.embeds[]	Усі вкладення, тобто об'єкти, що відповідають контейнерам <EMBED>
document.forms[]	Усі форми, тобто об'єкти, що відповідають контейнерам <FORM>
document.images[]	Усі малюнки, тобто об'єкти, що відповідають контейнерам <IMG>
document.links[]	Усі номери, тобто об'єкти, що відповідають контейнерам <A HREF="..."> і <AREA HREF="...">
document.f.elements[]	Усі елементи форми з ім’ям f - тобто об'єкти, що відповідають контейнерам <INPUT> і <SELECT>
document.f.s.options[]	Усі опції (контейнери <OPTION>) в контейнері <SELECT NAME=s> у формі <FORM NAME=f>
navigator.mimeTypes[]	Усі типи MIME, підтримувані браузером (перелік на сайті IANA)
function_name.arguments[]	Усі аргументи, передані функції function_name() у разі виклику

Властивості

Більшість HTML-контейнерів мають атрибути. Як вже відомо, кожному контейнеру відповідає об'єкт. Отже, відповідним атрибутам відповідають властивості об'єкта. Відповідність між атрибутами HTML-контейнерів і властивостями DOM-об'єктів не завжди пряма.

Зазвичай кожному атрибуту відповідає певна властивість об'єкта. Але, по-перше, назву цієї властивості не завжди легко зрозуміти за назвою атрибута, а по-друге, у об'єкта можуть бути властивості, що не

мають аналогів серед атрибутів. Крім того, атрибути є регістрозалежні або регістронезалежні, як і вся мова HTML, тоді як властивості об'єктів потрібно писати в точно визначеному регістрі символів.

Наприклад, контейнер якоря <A...>... </A> має атрибут HREF, який перетворює його в гіпертекстове посилання:

<A HREF="http://www.ikt.hneu.net.ua/"**>СДО ХНЕУ</A>**

Даному гіперпосиланню відповідає об'єкт (класу URL) – document.links [0], якщо припускати, що це перше посилання в нашому документі. Тоді атрибуту HREF відповідатиме властивість href цього об'єкта. До властивості об'єкта можна звертатися за допомогою точкової нотації: об'єкт.властивість. Наприклад, щоб змінити адресу, на яку вказує це посилання, можна написати:

document**.**links**[**0**].**href**=**'http://ya.ru/'**;.**

До властивостей можна також звертатися за допомогою дужкової нотації: об'єкт ['властивість']. Наприклад:

document**.**links**[**0**][**'href'**]=**'http://ya.ru/'**;.**

У об'єктів, що відповідають гіперпосиланнями, також є властивості, які не мають аналогів серед атрибутів. Наприклад, властивість document.links [0].protocol у наведеному прикладі дорівнює "http:" тощо. Повний перелік властивостей об'єктів класу URL розміщений в довіднику з JavaScript.

Методи

У термінології JavaScript методи об'єкта визначають функції, за допомогою яких виконуються дії з цим об'єктом, наприклад: зміна його властивостей, відображення їх на веб-сторінці, відсилання даних на сервер, перезавантаження сторінки тощо.

Якщо є посилання <A HREF="http://www.ikt.hneu.net.ua/"> СДО ХНЕУ </A> (вважається, воно є першим у нашому документі), то у відповідного до нього об'єкта document.links [0] є метод click(). Його

виклик в будь-якому місці JavaScript-програми рівносильний тому, якби користувач клікнув по посиланню, що демонструє приклад:

```
<A HREF="http://www.ikt.hneu.net.ua/">СДО ХНЕУ</A>
<SCRIPT> document.links[0].click(); </SCRIPT>
```

З відкриттям такої сторінки користувач одразу буде перенаправлений на сайт СДО ХНЕУ. Слід зауважити, що скрипт написаний після посилання. Якби його написали до посилання, то, оскільки в цей момент посилання (а отже, й об'єкта) ще не існує, браузер видав би повідомлення про помилку.

Деякі методи можуть застосовуватися неявно. Для всіх об'єктів визначено метод перетворення в рядок символів: toString(). Наприклад, при додаванні числа та рядка число буде перетворено на рядок:

```
"25"+5 = "25"+(5).toString() = "25"+"5" = "255"
```

Аналогічно, якщо звернутися до об'єкта window.location (розглянутому в наступній лекції) в строковому контексті, скажімо, всередині виклику document.write(), то неявно буде виконано це перетворення, і програміст цього не помітить, ніби він роздруковував не об'єкт, а рядок:

```
<SCRIPT>
  document.write('Неявне перетворення: ');
  document.write(window.location);
  document.write('<BR>Явне перетворення: ');
  document.write(window.location.toString());
</SCRIPT>
```

Цей ефект можна спостерігати для вбудованих об'єктів типу Date:

```
<SCRIPT>
  var d = new Date();

  document.write('Неявне перетворення: ');
  document.write(d);
  document.write('<BR>Явне перетворення: ');
  document.write(d.toString());
</SCRIPT>
```

Події

Окрім методів і властивостей, об'єкти характеризуються подіями. Власне, сутність програмування на JavaScript полягає в написанні обробників цих подій. Наприклад, з об'єктом типу *button* (контейнер INPUT типу *button* – "кнопка") може відбуватися подія **Click**, тобто користувач може натиснути на кнопку. Для цього атрибути контейнера INPUT розширені атрибутом обробляння цієї події – **onClick**. Як значення цього атрибута вказується програма обробляння події, яку повинен написати на JavaScript автор HTML-документа:

<INPUT TYPE=button VALUE="Натиснути"
onClick="alert('Будь ласка, натисніть ще раз')"**>**

Обробники подій вказуються в спеціально створених для цього атрибутах у тих контейнерах, з якими ці події пов'язані. Наприклад, контейнер BODY визначає властивості всього документа, тому обробник події "Завершено завантаження всього документа" вказується в цьому контейнері як значення атрибута **onLoad**.

Приклади подій: натискання користувачем кнопки у формі; установка фокусу в поле форми чи відведення фокуса з неї; зміна введеного в поле значення; натискання кнопки миші; відпускання кнопки миші; клацання кнопкою миші на об'єкті (посиланні, полі, кнопці, зображенні тощо); подвійне клацання кнопкою миші на об'єкті; переміщення покажчика миші; виділення тексту в полі вводу або на сторінці та інші. Однак деякі зміни, що відбуваються на сторінці, не генерують жодних подій; наприклад: зміна значення в полі вводу не користувачем, а скриптом; зміна фону документа; зміна (скриптом) значення атрибута HREF посиланням, а також зміна більшості інших атрибутів HTML-контейнерів. Про всі важливі події та про їх "перехоплення" розповідатиметься у відповідних лекціях.

Різні браузери можуть вести себе по-різному у випадку виникнення подій. Наступний приклад дозволяє визначити, в якому порядку викликаються обробники подій при кліці або подвійному кліці миші на засланні або кнопці (рис. 5.2).

```
<SCRIPT>
        function show_MouseDown(){
       rrr.innerHTML+='мишу натиснули (MouseDown)<br>';
        }
        function show_Click(){
       rrr.innerHTML+='клік миші (Click)<br>';
        }
        function show_MouseUp() {
       rrr.innerHTML+=' мишу відпустили (MouseUp)<br>';
        }
        function show_DblClick(){
       rrr.innerHTML+='подвійний клік (DblClick)<br>';
        }
       </SCRIPT>

       <A HREF="javascript:void(0);"
       onMouseDown="show_MouseDown();"
       onClick="show_Click();"
       onMouseUp="show_MouseUp();"
onDblClick="show_DblClick();">Посилання</A>

       <INPUT TYPE=button VALUE="Кнопка"
       onMouseDown="show_MouseDown();"
       onClick="show_Click();"
       onMouseUp="show_MouseUp();"  onDblClick="show_DblClick();">
       <BR>
<SPAN ID="rrr"></SPAN>
```

Рис. 5.2. **Стеження за подіями Click і DblClick**

У разі одиночного кліку на засланні або кнопці події виникають в порядку: **MouseDown, MouseUp, Click**, що логічно. Під час подвійного кліку послідовність подій, що відбуваються в різних браузерах, різна (рис. 5.3).

	**MouseDown** –> **MouseUp** –> **Click** –> *MouseDown* –> **MouseUp** –> *Click* –> **DblClick**
	MouseDown –> MouseUp –> Click –> MouseUp –> DblClick

Рис. 5.3. **Порівняння подій у разі подвійного натискання в браузерах**

Користувальницькі об'єкти

У даному розділі розглядаються три основні моменти:

поняття об'єкта;

прототип об'єкта;

методи об'єкта Object.

Поняття користувальницького об'єкта

Доцільно розглянути приклад (рис. 5.4) певного об'єкта класу Rectangle.

```
function Rectangle(a,b,c,d)
        {
          this.x0 = a;
          this.y0 = b;
          this.x1 = c;
          this.y1 = d;

          this.area = new Function(
        "return Math.abs((this.x1-this.x0)*(this.y1-this.y0))");
        }

r = new Rectangle(0,0,30,50);
```

Рис. 5.4. **Користувальницький об'єкт**

Функція Rectangle() – це конструктор об'єкта класу Rectangle, визначеного користувачем. Конструктор дозволяє створити екземпляр (об'єкт) даного класу, адже функція – це тільки опис певних дій. Для того щоб ці дії були виконані, необхідно передати функції управління. У

наведеному прикладі це здіснюється за допомогою оператора **new** Rectangle. Він викликає функцію Rectangle() і тим самим генерує реальний об'єкт **r**.

У результаті створюється чотири змінних: x0, y0, x1, y1 – властивості об'єкта r. До них можна отримати доступ тільки в контексті об'єкта даного класу, наприклад:

```
up_left_x = r.x0;
up_left_y = r.y0;
```

Крім властивостей, всередині конструктора Rectangle визначають об'єкт area класу Function(), застосувавши вбудований конструктор мови JavaScript. Це методи об'єкта класу Rectangle. Викликати цю функцію можна теж тільки в контексті об'єкта класу Rectangle:

```
sq = r.area();
```

Таким чином, *об'єкт* – це сукупність властивостей і методів, доступ до яких можна отримати, тільки створивши за допомогою конструктора об'єкт даного класу і використавши його контекст.

На практиці досить рідко доводиться мати справу з об'єктами, створеними програмістом. Справа в тому, що об'єкт створюється функцією-конструктором, яка визначається на конкретній сторінці. Отже, все, що створюється в рамках даної сторінки, не може бути успадковано іншими сторінками. Потрібні дуже вагомі підстави, щоб автор веб-вузла зайнявся розробкою бібліотеки користувальницьких класів об'єктів. Набагато простіше писати функції для кожної сторінки

Прототип

Зазвичай розробники використовують вбудовані об'єкти JavaScript (крім ієрархії об'єктів DOM), такі, як: **Data, Array і String**. У цьому сенсі цікавою є властивість об'єктів під назвою prototype. *Прототип* – це інша назва конструктора об'єкта конкретного класу. Наприклад, якщо потрібно додати метод до об'єкта класу **String**, то можна це зробити, як показано на рис. 5.5.

Є один істотний нюанс: новими методами та властивостями будуть володіти тільки ті об'єкти, які породжуються після зміни прототипу

об'єкта. Усі вбудовані об'єкти створюються до того, як JavaScript-програма отримає управління, що істотно обмежує застосування властивості **prototype**.

Ключове слово "this"

Ключове слово this в JavaScript працює своєрідно – не так, як в інших мовах. На відміну від PHP, Java, C ++ й інших, значення this в JavaScript не прив'язується статично до жодного об'єкта, а залежить від контексту виклику.

```
String.prototype.out = new
Function("a", "a.write(this)");

var s = "Привіт!";

s.out(document);

// Буде виведено: Привіт!
```

А

```
function Emphatic(bStrong){
var sEmTag = "em";
var sStrongTag = "strong";
var txt = this.toString();
var sTag =
(bStrong) ? sStrongTag : sEmTag;
return "<" + sTag + ">" + txt + "</" + sTag + ">";
        }

String.prototype.em = Emphatic;
…
document.write("Ми виділ\яємос\я!".em(false));
document.write("Ми дуже   виділ\яємос\я!".em(true));
```

Б

Рис. 5.5. **Використання прототипа**

Розберемо чотири можливих випадки.

Режим конструктора

Якщо функція викликається через **new** як конструктор об'єкта, то **this** ставиться на створюваний об'єкт (рис. 5.6).

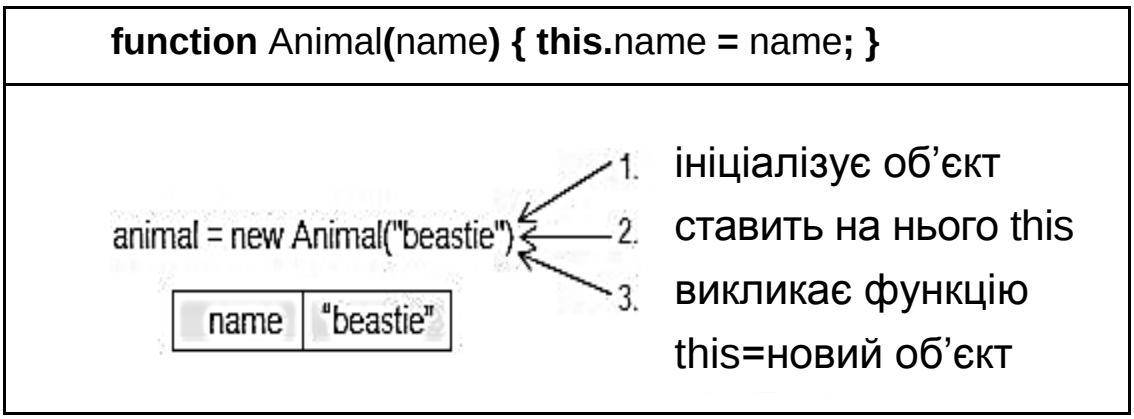


Рис. 5.6. **Використання "this" в режимі конструктора**

У результаті отримується об'єкт, ініціалізований конструктором, із заповненою властивістю name.

Метод об'єкта

Якщо функція запущена як властивість об'єкта, то в **this** буде посиланням на цей об'єкт.

```
obj.func(параметри)
// або
obj['func'](параметри)
```

При цьому абсолютно неважливо, звідки ця функція береться. Важливо лише, який об’єкт стоїть перед func. Приклад наведено на рис. 5.7. В обох випадках запускається одна і та ж функція say, яка отримує в **this** об'єкт до точки.

```
// Створимо два будь-яких об'єкта:
vasya = {
   name: "Василь"
}
dima = {
   name: "Дмитро"
}

// Визначимо не пов'язану з ними функцію say:
say = function() {
   alert("Привіт, я "+this.name)
}

// Надамо функцію властивості sayHi для обох об'єктів:
vasya.sayHi = say
dima.sayHi = say

// І тепер тестовий запуск:
vasya.sayHi()    // =>  "Привіт, я Василь"
dima.sayHi()    // => "Привіт, я Дмитро"
// или так
dima['sayHi']()
```

Рис. 5.7. **Використання "this" у методі об'єкта**

Apply/Call

Функцію можна викликати, використовуючи методи **call** і **apply**.

func.apply(**obj, [**параметри**])**
func.call(**obj,** параметр**)**

За такого виклику **this** буде встановлений в **obj**.

Обидва методи працюють однаково, але для **call** аргументи підставляються у виклик, а для **apply** – передаються в масив (рис. 5.8).

```
function sum(a,b) {
   this.c = a + b
}

var obj = {}

sum.call(obj, 1,2)
// або
sum.apply(obj, [1,2])

alert(obj.c)   // => 3
```

Рис. 5.8. **Використання "this" у функції стосовно об'єкта**

Виклик через apply зручний тим, що можна самостійно сформувати масив аргументів або модифікувати існуючий.

Найпростіший виклик **say()**.

При такому виклику **this** ставиться рівним глобальному об'єкту (в браузері це **window**). Зазвичай, якщо функція використовує **this**, вона не має на увазі запуск в такому вигляді, тому в 90 % випадків таке значення **this** – результат помилки в програмі.

Об'єкт window

Клас об'єктів Window – це найстарший клас в ієрархії об'єктів JavaScript. Об'єкт window, що відноситься до поточного вікна (тобто в якому виконується скрипт), є об'єктом класу Window. Клас об'єктів Frame міститься в класі Window, тобто кожен фрейм – це теж об'єкт класу Window.

Об'єкт window створюється тільки в момент відкриття вікна. Усі інші об'єкти, які породжуються під час завантаження сторінки, є властивостями об'єкта window. Більш того, всі глобальні змінні, визначені в даному вікні, теж є властивостями об'єкта window. Таким чином, у об'єкта window можуть бути різні властивості під час завантаження різних сторінок. Крім того, в різних браузерах властивості об'єктів і поведінка об'єктів і браузера при оброблянні подій може бути різним. Під час програмування на JavaScript найчастіше використовують такі властивості, методи і події об'єкта window (табл. 5.2).

Таблиця 5.2

Властивості, методи та події об'єкта window

Властивості	Методи	Події
status	open()	Load
defaultStatus	close()	Unload
location	focus()	Focus
history	blur()	Blur
navigator		
	alert()	Resize
document	confirm()	Error
frames[]	prompt()	
opener	setTimeout()	
parent	setInterval()	
self	clearTimeout()	
top	clearInterval()	

Оскільки об'єкт window є найстаршим, то в більшості випадків, звертаючись до його властивостей і методів, приставку "window." можна опускати (зрозуміло, в разі, якщо треба звернутися до властивості або методу поточного вікна, де працює скрипт; якщо ж це інше вікно, то необхідно вказати його ідентифікатор). Так, наприклад, можна писати alert ('Привіт') замість window.alert ('Привіт'), або location замість window.location. Винятками з цього правила є виклики методів open() і close(), у яких потрібно вказувати ім’я вікна, з яким працюють (батьківське в першому випадку та дочірнє – у другому).

Властивості об'єкта window

Поле статусу та властивість ***window.status***

Поле статусу – це перше, що почали використовувати автори HTML-сторінок з арсеналу JavaScript. Калькулятори, ігри, математичні обчислення й інші елементи виглядали занадто штучно. На їх фоні рухомий рядок в полі статусу був родзинкою, яка могла дійсно привернути увагу користувачів до Web-вузла. Поступово його популярність зійшла нанівець. Рухомі рядки стали рідкістю, але програмування поля статусу зустрічається на багатьох веб-вузлах.

Полем статусу (status bar) називають поле в нижній частині вікна браузера одразу під областю відображення HTML-сторінки. У полі статусу відображується інформація про стан браузера (завантаження документа, завантаження графіки, завершення завантаження, запуск аплета тощо). Програма на JavaScript має можливість працювати з цим полем як із змінним властивістю вікна. При цьому фактично з ним пов'язані дві відмінних властивості:

window.status – значення поля статусу;

window.defaultStatus – значення поля статусу за замовчуванням.

Значення властивості status можна змінити – і вона тут же буде відображена в полі статусу. Властивість defaultStatus теж можна міняти – і оразу по зміні вона відображується в полі статусу.

Відмінність між цими двома властивостями полягає в їх поведінці: якщо до властивості status додати порожній рядок: window.status = "", то в полі статусу автоматично буде відображено значення defaultStatus. Зворотного ж не відбувається: якщо привласнені порожні рядки властивості defaultStatus, вона відобразиться в полі статусу незалежно від значення властивості status. Слід зазначити, що реакція браузерів на описувані нижче дії з властивостями status і defaultStatus може бути різною в різних браузерах.

Програмування ***status***

Властивість status пов'язана з відображенням повідомлень про події, відмінним від простого завантаження сторінки. Наприклад, в Internet Explorer наведенням покажчика миші на посилання обробник onMouseOver поміщає в поле статусу значення URL, вказане в атрибуті HREF цього посилання (не змінюючи значення властивостей status і defaultStatus). У разі попадання курсора миші на область, вільну від посилань, обробник onMouseOut повертає в поле статусу значення defaultStatus за умови, що це не порожній рядок (знову не змінюючи значень обох властивостей). Можна змінити цю поведінку, наприклад, як на рис. 5.9.

```
<A onMouseOver="window.status='Миша над посиланням ';return true;"
        onMouseOut="window.status='Мишу повели з посилання ';"
        HREF="http://site.com/"> Наведіть мишу на посилання та
слідкуйте за полем статусу </A>
```

Рис. 5.9. **Програмування status**

Слід звернути увагу на оператор return true наприкінці обробника подій onMouseOver. Він необхідний для того, щоб скасувати дію за замовчуванням (у даному випадку – висновок URL у полі статусу), яку, за відсутності цього оператора, браузер виконав би одразу після виводу свого рядка в поле статусу, і користувач не встиг би побачити рядок. Аналогічне скасування дії за замовчуванням слушне і для деяких інших подій (onClick, onKeyDown, onKeyPress, onMouseDown, onMouseUp, onSubmit, onReset) з тією лише різницею, що для перелічених обробників відміна виконується оператором return false.

На жаль, для обробника onMouseOut такого способу скасування дії за замовчуванням не існує. Але в даному конкретному випадку це не потрібно: як уже було сказано, при відведенні курсора з посилання в полі статусу відновлюється значення defaultStatus тільки в разі, якщо це значення не є порожнім рядком. Але в даному випадку (за замовчуванням, під час завантаження сторінки в IE) воно дорівнює саме порожньому рядку. Тому, ведучи курсор з нашого лінку, в полі статусу залишається рядок "Мишу повели з посилання". Ситуація зміниться в наступному прикладі, якщо попередньо задати своє (непорожнє) значення defaultStatus.

*Програмування **defaultStatus***

Властивість defaultStatus визначає текст, що відображується в полі статусу, коли жодних подій не відбувається. Попередній приклад слід доповнити зміною цієї властивості в момент закінчення завантаження документа, тобто в обробнику onLoad (рис. 5.10).

```
<BODY onLoad="window.defaultStatus='Значення за замовчуванням';">

<A onMouseOver="window.status='Миша над посиланням '; return true;"
onMouseOut="window.status=' Мишу повели з посилання '; alert('Ждем');"
HREF="http://site.com/"> Наведіть мишу на посилання та слідкуйте за полем статусу </A>
</BODY>
```

Рис. 5.10. **Програмування defaultStatus**

Одразу після завантаження документа в полі статусу відобразиться "Значення за умовчанням". У разі наведення покажчика миші на посилання в полі статусу з'явиться напис "Миша над посиланням", при цьому URL посилання (http://site.com/) в полі статусу не з'явиться, тому що його висновок скасований оператором return true.

Якщо прибрати покажчик миші з посилання, то користувач не встигне побачити рядок "Мишу повели з посилання", оскільки дія за замовчуванням (висновок значення defaultStatus у полі статусу) не пригнічується (і не може бути скасована – у обробника onMouseOut немає такої можливості). Однак якщо ввести оператор виводу вікна попередження alert ('Чекаємо'), то тепер користувач бачитиме в полі статусу рядок "Мишу повели з посилання", поки не натисне OK на цьому вікні.

Поле адреси та властивість ***window.location***

Поле адреси в браузері зазвичай розташовується у верхній частині вікна та відображує URL завантаженого документа. Якщо користувач хоче вручну перейти до якої-небудь сторінки (набрати її URL), він робить це в полі адреси.

Властивість location об'єкта window є об'єктом класу Location. Клас Location, в свою чергу, є підкласом класу URL, до якого належать також об'єкти класів Area та Link. Об'єкти Location успадковують усі властивості об'єктів URL, що дозволяє отримати доступ до будь-якої частини схеми URL.

У цілях сумісності з попередніми версіями JavaScript у мові підтримується також властивість window.document.location, яка повністю дублює window.location з усіма його властивостями та методами. Наразі слід розглянути властивості та методи об'єкта window.location (подій, пов'язаних з цим об'єктом, нема).

*Властивості об'єкта **location***

Їх простіше продемонструвати на прикладі. Якщо браузер відображує сторінку, розташовану за адресою:

http:***//***www***.site.ru***:***80/***dir***/***page***.***cgi***?***product***=***phone***&***id***=3***#mark

тоді властивості об'єкта location отримують наступні значення:

window.location.href =
"http://www.site.ru:80/dir/page.cgi?product=phone&id=3#mark"
window.location.protocol = "http:"
window.location.hostname = "www.site.ru"
window.location.port = 80
window.location.host = "www.site.ru:80"
window.location.pathname = "dir/page.cgi"
window.location.search = "?product=phone&id=3"
window.location.hash = "#mark"

Як вже говорилося в попередніх лекціях, до властивостей об'єктів можна звертатися як за допомогою точкової нотації (як вище), так і за допомогою дужкової нотації, наприклад: window.location ['host'].

*Методи об'єкта **location***

Методи об'єкта location призначені для управління завантаженням і перезавантаженням сторінки. Це управління полягає в тому, що можна перезавантажити поточний документ (метод reload()) або завантажити новий (метод replace()).

window.location.reload (true);

Метод reload() повністю моделює поведінку браузера натисканням на кнопку Reload на панелі інструментів. Якщо викликати метод без аргументу або вказати його рівним true, то браузер перевірить час останньої модифікації документа та завантажить його з кешу (якщо документ не був модифікований) або із сервера. Така поведінка відповідає простому натисканню кнопки Reload браузера (клавіші F5 в Internet Explorer). Якщо в якості аргументу вказати false, то браузер обов’язково перезавантажить поточний документ з сервера. Така поведінка відповідає одночасному натисканню клавіші Shift і кнопки браузера Reload (або Ctrl + F5 в Internet Explorer).

Використовуючи об'єкт location, перейти на нову сторінку можна двома способами:

```
window.location.href="http://www.newsite.ru/";
window.location.replace("http://www.newsite.ru/");
```

Відмінність між ними – у відображенні цієї дії в історії відвідувань сторінок window.history. У першому випадку в історію відвідувань додасться новий елемент, що містить адресу "http://www.newsite.ru/", тому при бажанні можна буде натиснути кнопку Back на панелі браузера, щоб повернутися до попередньої сторінки. У другому випадку нову адресу "http://www.newsite.ru/" замістить попередня з історії відвідувань, і повернутися до попередньої сторінки натисканням кнопки Back вже не можна.

Історія відвідувань ***history***

Історія відвідувань сторінок World Wide Web дозволяє користувачеві повернутися до сторінки, яку він переглядав раніше в даному вікні браузера. Історія відвідувань в JavaScript трансформується в об'єкт window.history. Цей об'єкт вказує на масив URL-сторінок, які користувач відвідував і які він може отримати, вибравши з меню браузера режим Go. Методи об'єкта history дозволяють завантажувати сторінки, використовуючи URL з цього масиву.

Щоб не виникло проблем з безпекою браузера, подорожувати по History можна, тільки використовуючи індекс. При цьому URL як текстовий рядок програмісту недоступний. Найчастіше цей об'єкт використовують у прикладах або сторінках, на які можуть бути посилання

з декількох різних сторінок, припускаючи, що можна повернутись до сторінки, з якої приклад буде завантажений:

```
<FORM> <INPUT TYPE="button" VALUE="Назад"
onClick="history.back()" /> </FORM>
```

Даний код відображує кнопку "Назад", натиснувши на яку, повертається попередня сторінка. Аналогічним чином діє метод history.forward(), переносячи користувача на наступну сторінку.

Існує також метод go(), який має цілочисельний аргумент і дозволяє переходити на кілька кроків вперед або назад по історії відвідувань. Наприклад, history.go(-3) перенесе на три кроки назад в історії перегляду. При цьому методи back() і forward() рівнозначні методу go() з аргументами -1 і 1, відповідно. Виклик history.go(0) призведе до перезавантаження поточної сторінки.

Тип браузера **navigator**

Часто виникає завдання налагодження сторінки на конкретну програму перегляду (браузер). При цьому можливі два варіанти: визначення типу браузера на боці сервера або на боці клієнта. Для останнього варіанту в арсеналі об'єктів JavaScript існує об'єкт window.navigator. Найважливіші з властивостей цього об'єкта наведені в табл. 5.3.

Таблиця 5.3

Основні властивості об'єкта window.navigator

Властивості	Опис
userAgent	Основна інформація про браузер. Передається серверу в HTTP-заголовку в разі відкриття користувачем сторінок
appName	Назва браузера
appCodeName	Кодова назва браузера
appVersion	Дані про версію браузера та сумісності

Простий приклад визначення типу програми перегляду:

```
<FORM> <INPUT TYPE=button VALUE="Тип навігатора"
onClick="alert(window.navigator.userAgent);"> </FORM>
```

Після натискання на кнопку відображується вікно попередження, яке містить значення властивості **navigator.userAgent**. Якщо це значення розібрати за компонентами, то може вийти, наприклад, наступне:

navigator**.userAgent = "**Mozilla**/4.0 (**compatible; MSIE8.0; Windows NT 6.1; Trident/4.0; SLCC2;.NET CLR 2.0.50727;.NET CLR 3.5.30729;.NET CLR 3.0.30729; Media Center PC 6.0)**"**

navigator.*appName* = "Microsoft Internet Explorer"
navigator.*appCodeName* = "Mozilla"
navigator.appVersion = "4.0 (compatible; MSIE8.0; Windows NT 6.1; Trident/4.0; SLCC2;.NET CLR 2.0.50727;.NET CLR 3.5.30729;.NET CLR 3.0.30729; Media Center PC 6.0)"

У об'єкта navigator є ще кілька цікавих з позицій програмування застосувань. Наприклад, щоб перевірити, чи підтримує браузер клієнта мову Java, треба застосувати метод navigator.javaEnabled(), який повертає та підтримує значення true і false – у протилежному випадку.

Можна перевірити, які формати графічних файлів підтримує браузер (рис. 5.11), скориставшись властивістю navigator.mimeTypes (вона є масивом всіх типів MIME, які підтримуються даним браузером).

```
<SCRIPT>
  if(navigator.mimeTypes['image/gif']!=null)
    document.write('Ваш браузер підтримує GIF<BR>');
  if(navigator.mimeTypes['image/tif']==null)
    document.write('Ваш браузер не підтримує TIFF');
</SCRIPT>
```

Рис. 5.11. **Перевірка підтримки MIME-типів браузером**

*Методи об'єкта **window***

Які операції можна виконувати з вікном? Відкрити (створити), закрити (видалити), покласти його поверх всіх інших відкритих вікон (передати фокус). Крім того, можна управляти властивостями вікна та

властивостями підлеглих йому об'єктів. Слід зосередитись на простих і найбільш популярних методах управління вікнами.

alert()

Метод alert() дозволяє створити вікно попередження, яке має тільки кнопку "OK":

```
<A HREF="javascript:window.alert('Увага')">
Повторіть запит!</A>
```

Потрібно мати на увазі, що повідомлення виводяться системним шрифтом, отже, для отримання попереджень російською мовою потрібна локалізована версія ОС.

confirm()

Метод confirm() дозволяє задати користувачеві питання, на яке можна відповісти позитивно (натиснувши кнопку "OK") або негативно (натиснувши кнопку "Скасування" або "Cancel" чи просто закривши вікно запиту). Відповідно до дій користувача метод confirm() повертає значення true або false (рис. 5.12).

```
<FORM NAME=f>
        <INPUT TYPE=button NAME=b VALUE="Натисніть цю кнопку"
        onClick="if(window.confirm('Ви знаєте JavaScript?'))
            document.f.b.value='Так. Запитати ще?';
        else document.f.b.value='Ні. Запитати ще?';">
</FORM>
```

Рис. 5.12. **Метод confirm()**

Усі обмеження для повідомлень російською мовою, які були описані для методу alert(), справедливі і для методу confirm().

prompt()

Метод prompt() дозволяє прийняти від користувача рядок тексту. Синтаксис його такий:

```
prompt("Рядок питання","Рядок відповіді за замовчуванням")
```

Коли користувач введе свою відповідь (або залишить незмінним відповідь за замовчуванням) і натисне кнопку OK, метод prompt() поверне отриманий рядок у якості значення, яке може далі набути будь-яка змінна і яке можна в подальшому розбирати в JavaScript-програмі (рис. 5.13).

```
<FORM NAME=f>
        <INPUT TYPE=button VALUE=" Відкрити вікно вводу "
        onClick="document.f.e.value=
           window.prompt('Введіть повідомлення', 'Сюди');">
        <INPUT SIZE=30 NAME=e>
</FORM>
```

Рис. 5.13. **Метод prompt()**

open()

Метод open() призначений для створення нових вікон. У загальному випадку його синтаксис виглядає наступним чином:

```
myWin = window.open("URL", " ім'я_вікна ",
" параметр = значення, параметр = значення,... ",
замінити);
```

Перший аргумент задає адресу сторінки, яка завантажується в нове вікно (можна залишити порожній рядок, тоді вікно залишиться порожнім). Другий аргумент задає ім’я вікна, яке можна буде використовувати в атрибуті TARGET контейнерів <A> і <FORM>. Як значення допустимі також зарезервовані імена _blank, _parent, _self, _top, сенс яких такий же, як у аналогічних значень атрибута TARGET. Якщо ім’я_вікна збігається з ім’ям вже існуючого вікна (або кадру), то нове вікно не створюється, а всі наступні маніпуляції зі змінною myWin застосовуватимуться до цього вікна (або фрейму).

Третій аргумент не містить пробілів у рядках і є списком параметрів і їх значень, поданих через кому. Вказівка на кожний з параметрів необов'язкова, однак значення за замовчуванням можуть залежати від

браузера, тому завжди треба вказувати саме ті параметри, які очікуються. Можливі параметри перераховані в табл. 5.4. Замість значень yes і no можна використовувати 1 і 0. Останній аргумент "Замінити" є необов'язковим, бо приймає значення true і false та визначає, слід новий URL додати до history в якості нового елемента чи замінити ним останній елемент history.

Метод window.open() повертає посилання на знову відкрите вікно, тобто об'єкт класу Window. Його можна надати до змінної (що й було зроблено вище), з тим щоб надалі можна було управляти відкритим вікном (писати в ньому, читати з нього, передавати та прибирати фокус, закривати).

Таблиця 5.4

Параметри методу window.open()

Параметр	Значення	Опис
width	число	Ширина вікна в пікселях (не менше 100)
height	число	Висота вікна в пікселях (не менше 100)
left	число	Відстань від лівого краю екрану до лівої межі вікна в пікселях
top	число	Відстань від верхнього краю екрана до верхньої межі вікна в пікселях
directories	yes/no	Наявність у вікна панелі папок (Netscape Navigator)
location	yes/no	Наявність у вікна поля адреси
menubar	yes/no	Наявність у вікна панелі меню
resizable	yes/no	Чи зможе користувач змінювати розмір вікна
scrollbars	yes/no	Наявність у вікна смуг прокрутки
status	yes/no	Наявність у вікна поля статусу
toolbar	yes/no	Наявність у вікна панелі інструментів

Два приклади відкриття нового вікна, зображені на рис. 5.14.

```
<FORM>
        <INPUT TYPE=button VALUE=" просте вікно "
         onClick="window.open('', 'test1',
          'directories=no,height=200,location=no,'+
          'menubar=no,resizable=no,scrollbars=no,'+
          'status=no,toolbar=no,width=200');">

        <INPUT TYPE=button VALUE="Складне вікно"
         onClick="window.open('', 'test2',
          'directories=yes,height=200,location=yes,'+
          'menubar=yes,resizable=yes,scrollbars=yes,'+
          'status=yes,toolbar=yes,width=200');">
</FORM>
```

Рис. 5.14. **Приклади відкриття нового вікна**

close()

Метод **close()** дозволяє закрити вікно. Найчастіше виникає питання, яке з вікон, власне, слід закрити. Якщо необхідно закрити поточне, то:

```
window.close();
self.close();
```

Якщо вікно відкрите за допомогою методу window.open(), то зі скрипта, працюючого в новому вікні, послатися на вікно-батько можна за допомогою window.opener (тут window посилається на об'єкт нового, створеного вікна, оскільки воно використано в скрипті, працюючому в новому вікні). Тому, якщо необхідно закрити батьківське вікно, тобто вікно, з якого було відкрито поточне, то:

```
window.opener.close();
```

Якщо необхідно закрити вільне вікно, то спочатку потрібно отримати його ідентифікатор:

```
id=window.open();
...
id.close();
```

Як видно з останнього прикладу, закривають вікно не за ім’ям (значення атрибута TARGET не враховується), а використовують покажчик на об'єкт.

focus() и blur()

Метод **focus()** застосовується для передавання фокуса в вікно, з яким він використовувався. Передавання фокуса корисне як під час відкриття вікна, так і за його закритті, не кажучи вже про випадки, коли потрібно вибирати вікна. Як приклад можна навести наступне.

Відкрити вікно і, не закриваючи його, знову відкрити вікно з таким же ім'ям, але з іншим текстом. Нове вікно не з'явилося поверх основного вікна, бо фокус йому не був переданий. Тепер повторити відкриття вікна, але вже з передачею фокуса (рис. 5.15).

Оскільки зміст нового вікна записується з вікна старого (батька), то в якості покажчика на об'єкт використовується значення змінної myWin.

Щоб відвести фокус з певного вікна myWin, необхідно застосувати метод myWin.blur().

Наприклад, щоб відвести фокус з поточного вікна, де виконується скрипт, потрібно викликати window.blur(). Ефект буде такий, ніби користувач сам згорнув вікно натисканням кнопки в правому верхньому кутку вікна.

```
<HTML>
        <HEAD>
         <SCRIPT>
          function myfocus(a){
           myWin = window.open('','example','width=300,height=200');
                 // відкриваємо вікно і заводимо змінну з покажчиком на нього.
               // Якщо вікно з ім'ям 'example' існує, то нове вікно не створюється,
                // А відкривається потік для запису в наявне вікно з ім'ям 'example'
            if(a==1){
            // відкриваємо потік введення в уже створене вікно
              myWin.document.open();
              // Пишемо в цей потік
            myWin.document.write('<H1> Відкрили вікно вперше ');
            }
            if(a==2){
              myWin.document.open();
              myWin.document.write('<H1> Відкрили вікно вдруге ');
            }
            if(a==3){
              myWin.focus(); // передаємо фокус, а потім виконуємо ті ж дії,
                                        // що і в попередньому випадку
           myWin.document.open();
              myWin.document.write('<H1>  Відкрили вікно втретє ');
            }
              myWin.document.write('</H1>');
              myWin.document.close();
           }
        </SCRIPT>
        </HEAD>
        <BODY>
            <a href="javascript:myfocus(1);"> Відкриємо вікно і напишемо в нього щось, </a> <BR>
            <a href="javascript:myfocus(2);"> напишемо в нього ж щось інше, але фокус не передамо;</a><BR>
            <a href="javascript:myfocus(3);"> знову щось напишемо в нього, але спершу передавши йому фокус.</a>
        </BODY>
</HTML>
```

Рис. 5.15. **Передавання фокуса в нове вікно**

setTimeout()

Метод **setTimeout()** використовується для створення нового потоку обчислень, виконання якого відкладається на час (в мілісекундах), вказаний іншим аргументом:

idt = setTimeout("JavaScript_код",Time);

Типове застосування цієї функції – організація періодичної зміни властивостей об'єктів. Наприклад, можна запустити годинник в поле форми (рис. 5.16).

```
<HTML>
        <HEAD>
          <SCRIPT>
            var Chasy_idut=false;
            function myclock(){
             if(Chasy_idut){
               d = new Date();
               document.f.c.value =
               d.getHours()+':'+
               d.getMinutes()+':'+
               d.getSeconds();
             }
             setTimeout("myclock();",500);
            }
            function FlipFlag(){
              Chasy_idut = !Chasy_idut;
              document.f.b.value = (Chasy_idut)?
               'Остановити' : 'Запустити';
            }
          </SCRIPT>
        </HEAD>
        <BODY onLoad="myclock();">
          <FORM NAME=f>
            Поточний час:<INPUT NAME=c size=8>
            <INPUT TYPE=button name=b VALUE="Запустити"
            onClick="FlipFlag();">
          </FORM>
        </BODY>
</HTML>
```

Рис. 5.16. **Годинники в полі форми з використанням setTimeout()**

Слід звернути увагу, що потік породжується, тобто викликається setTimeout(), завжди, навіть в тому випадку, коли зупиняється показ годин. Якби він створювався тільки зі значенням змінної Chasy_idut =

true, то годинник би просто не запустився, бо на самому початку виконання скрипта встановлено var Chasy_idut = false. Але навіть якби був встановлений на початку var Chasy_idut = true, то годинник би запустився при завантаженні сторінки, а після зупинки потік би зник і у разі подальшого натисненні кнопки "Запустити" годинник продовжував би стояти.

clearTimeout()

Метод **clearTimeout()** дозволяє знищити потік, викликаний методом setTimeout(). Очевидно, що його застосування дозволяє більш ефективно розподіляти ресурси обчислювальної установки. Для того щоб використовувати цей метод у прикладі з годинником, потрібно модифікувати функції та форму (рис. 5.17).

```
<HTML>
        <HEAD>
          <SCRIPT>
            var Chasy_idut=false;
            var potok;
            function StartClock(){
              d = new Date();
              document.f.c.value =
               d.getHours()+':'+
               d.getMinutes()+':'+
               d.getSeconds();
              potok = setTimeout('StartClock();',500);
              Chasy_idut=true;
            }
            function StopClock(){
              clearTimeout(potok);
              Chasy_idut=false;
            }
          </SCRIPT>
        </HEAD>
        <BODY>
          <FORM NAME=f>
            Поточний час:<INPUT NAME=c size=8>
      <INPUT TYPE=button  VALUE="Запустити"
onClick="if(!Chasy_idut) StartClock();">
      <INPUT TYPE=button  VALUE="Зупинити"
onClick="if(Chasy_idut) StopClock();">
          </FORM>
        </BODY>
</HTML>
```

Рис. 5.17. **Годинники з використанням setTimeout() і clearTimeout()**

У даному прикладі для зупинки годинника використовується метод clearTimeout(). При цьому, щоб не породжувалося безліч потоків, перевіряється значення покажчика на об'єкт потоку.

setInterval() і clearInterval()

У попередніх прикладах для того, щоб потік запускався знову і знову, у функцію в якості останнього оператора поміщався виклик методу **setTimeout**(). Однак в JavaScript для цих цілей є спеціальні методи. Метод **setInterval** ("код_JavaScript", time) виконує код_JavaScript з періодом раз в time мілісекунд. Значення, що повертається, – посилання на створений потік. Щоб зупинити потік, необхідно викликати метод **clearInterval** (потік).

*Події об'єкта **window***

Доцільно зупинитися на подіях, пов'язаних з об'єктом window. Обробники цих подій зазвичай поміщають як атрибут контейнера <BODY>.

Load

Load – подія відбувається в момент, коли завантаження документа в даному вікні повністю закінчене. Якщо поточним вікном є фрейм, то подія Load його об'єкта window відбувається, коли в даному фреймі завантаження документа закінчилось, незалежно від стану завантаження документів в інших фреймах. Використовувати обробник даної події можна, наприклад, наступним чином:

<BODY onLoad="alert('Документ повністю завантажений.');"**>**

Unload

Unload – подія відбувається в момент вивантаження сторінки з вікна. Наприклад, коли користувач закриває вікно або переходить з цієї веб-сторінки на іншу в разі посилання або набору адреси в адресному рядку чи у випадку зміни адреси сторінки (властивості window.location) скриптом. Наприклад, у разі виходу користувача з нашої сторінки можна подбати про його зручність і закрити відкрите раніше нашим скриптом вікно:

<BODY onUnload="myWin.close();"**>**

Error

Error – подія відбувається у разі виникнення помилки в процесі завантаження сторінки. Якщо ця подія відбулася, можна, наприклад, вивести користувачу повідомлення за допомогою alert() або спробувати перезавантажити сторінку за допомогою window.location.reload(). У наступному прикладі (рис. 5.18) обробником події Error призначено функцію ff(), яка видаватиме повідомлення. Якщо в тексті програми допущено помилку: слово Alert написано з великої літери (в JavaScript це неприпустимо), після відкриття цього прикладу виникне помилка, і користувач отримає про це "дружнє" повідомлення.

```
<SCRIPT>
   function ff()
  { alert('Виникла помилка. Зв’яжіться з Web-майстром.'); }
  window.onerror = ff;
  Alert('Привіт');
</SCRIPT>
```

Рис. 5.18. **Установка обробника помилок**

Focus

Focus – подія відбувається в момент, коли вікну передається фокус. Наприклад, коли користувач "розкриває" згорнуте раніше вікно або (у Windows) вибирає це вікно браузера за допомогою Alt + Tab серед вікон інших додатків. Ця подія відбувається також при програмному передаванні фокусу даних вікна шляхом виклику методу window.**focus**(). Приклад використання:

```
<BODY onFocus="alert('Дякую, що повернулися!');">
```

Blur

Blur – подія, протилежна попередній, відбувається в момент, коли дане вікно втрачає фокус. Це може статися в результаті дій користувача або програмними засобами – викликом методу window.**blur**().

Resize

Resize – подія відбувається під час зміни розмірів вікна користувачем або сценарієм.

Змінні як властивості вікна

Глобальні змінні насправді є властивостями об'єкта window. У наступному прикладі (рис. 5.19) відкривається вікно з ідентифікатором wid. У ньому оголошується глобальна змінна t і надалі використовується у вікні-батьку з посиланням на неї як wid.t.

```
<HTML>
        <HEAD>
          <SCRIPT>
            wid = window.open('','','width=750,height=100,status=yes');
            wid.document.open(); R = wid.document.write;
            R('<HTML><HEAD><SCRIPT>var t;<\/SCRIPT></HEAD>');
            R('<BODY><H1>Нове вікно</H1></BODY></HTML>');
            wid.document.close();
          </SCRIPT>
        </HEAD>
        <BODY>
           <A HREF="javascript:
                wid.t=window.prompt('Новий стан:','');
                wid.status=wid.t; wid.focus(); void(0);">
                Змінено значення змінної t у новому вікні</A>
        </BODY>
</HTML>
```

Рис. 5.19. **Зміна змінної відкритого вікна**

Зверніть увагу на нюанс: усередині скрипта написаний <\/SCRIPT>. Комбінація "\ /" видає на виході "/". Зроблено це для того, щоб браузер (точніше, його HTML-парсер) не сприйняв би </ SCRIPT> як завершальний тег нашого (зовнішнього) скрипта. Детальніше цей аспект обговорювався у вступній лекції. Також зверніть увагу на аліас (синонім) R, який надано методу wid.document.write, щоб мати можливість коротко викликати його як R (...).

Аналогічним чином (з приставкою wid, що вказує на об'єкт вікна) можна звертатися до всіх елементів, що знаходяться у відкритому нами вікні, наприклад до форм. Як приклад розглядається зміна поля вводу у

вікні-нащадку з вікна-предка (рис. 5.20). Дочірнє вікно створюється за допомогою функції okno(), в ньому створюється форма, далі слід звернутися до поля цієї форми з вікна-предка.

```
<HTML>
        <HEAD>
          <SCRIPT>
            var wid; // Оголошуємо глобальну змінну
            function okno()
            {
              wid = window.open('','okoshko','width=500,height=200');
              wid.document.open(); R = wid.document.write;
              R('<HTML><BODY><H1> Міняємо текст у вікні-нащадку:</H1>');
              R('<FORM NAME=f><INPUT SIZE=40 NAME=t VALUE=Текст>');
              R('</FORM></BODY></HTML>');
              wid.document.close();
            }
          </SCRIPT>
        </HEAD>
        <BODY>
          <INPUT TYPE=button VALUE=" Відкрити вікно прикладу "
onClick="okno()">
          <INPUT TYPE=button VALUE=" Написати поточний час в полі вводу "
              onClick="window.wid.document.f.t.value=new Date();
                  window.wid.focus();">
        </BODY>
</HTML>
```

Рис. 5.20. **Зміна поля статусу у відкритому вікні**

Відкриваючи вікно-нащадок, в змінну wid поміщається покажчик на вікно: wid = window.open(...);. Тепер можна використовувати wid як ідентифікатор об'єкта класу Window. Виклик методу window.wid.focus() у цьому випадку обов'язковий, оскільки після натискання на кнопку "Написати поточний час у полі введення" відбувається передавання фокуса в батьківське вікно (яке може заступати знову відкрите вікно, так що зміни у вікні-нащадку не будуть видні користувачеві). Для того щоб побачити зміни, потрібно передати фокус у вікно-нащадок.

Мінлива wid повинна бути глобальною, тобто визначена за межами будь-яких функцій (як зроблено в даному прикладі). У цьому випадку вона стає властивістю об'єкта window, тому слід звертатись до неї в

обробнику onClick за допомогою window.wid. Якби вона була розміщена всередині опису функції okno() та написане var wid = window.open (...), до неї не можна було б звернутися з обробника події onClick, що знаходиться поза функції okno().

Об'єктна модель документа (DOM)

Більшість дій в JavaScript виконується з HTML-сторінкою. У JavaScript сторінка подана у вигляді об'єктної моделі DOM (Document Object Model).

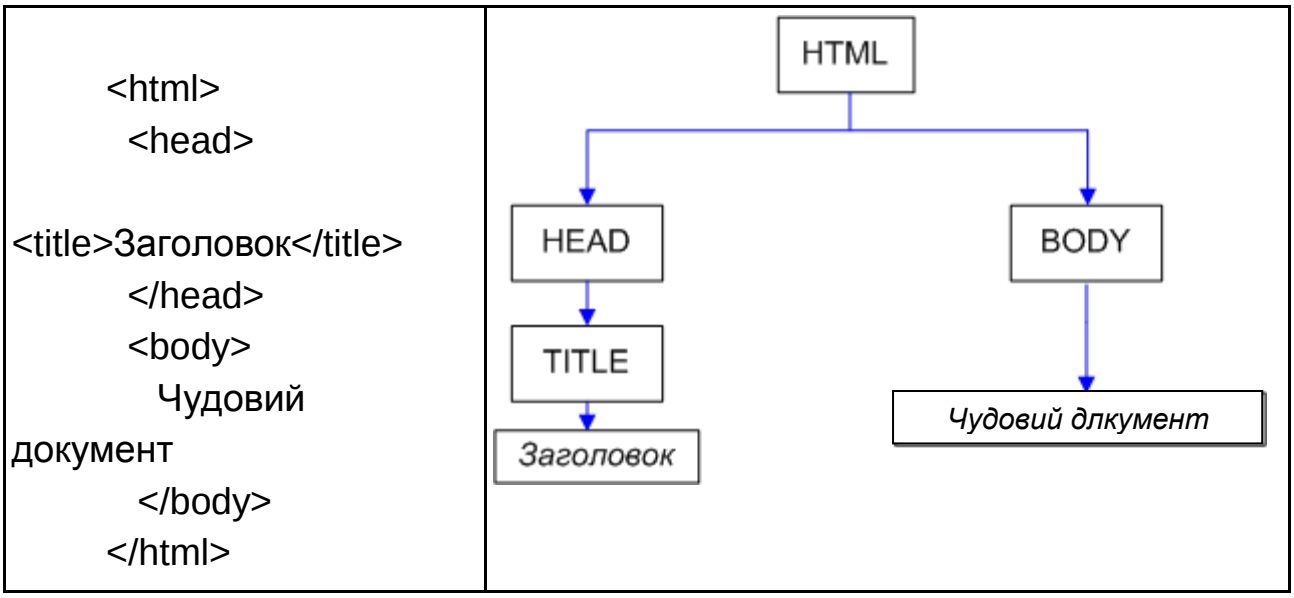
Будь-які дії зі сторінкою вимагають виклику відповідного методу DOM.

Ієрархічна модель DOM

Згідно з DOM-моделлю, документ є ієрархією.

Кожен HTML-тег утворює окремий елемент-вузол, кожен фрагмент тексту – текстовий елемент тощо.

Отже, DOM – це подання документа у вигляді дерева тегів. Це дерево (рис. 5.21) утворюється за рахунок вкладеної структури тегів плюс текстові фрагменти сторінки, кожен з яких утворює окремий вузол.



А) html Б) ієрархічна модель DOM

Рис. 5.21. **Найпростіша DOM**

Найбільш зовнішній тег – <html>, тому дерево починається від нього.

Усередині <html> знаходяться два вузли: <head> і <body>. Вони стають дочірніми вузлами для <html>.

Теги утворюють вузли-елементи (element node). Текст поданий текстовими вузлами (text node). Обидва узли – рівноправні вузли дерева DOM.

Більш насичену елементами html-сторінку зображено на рис. 5.22.

```
<html>
        <head>
          <title>
            Про лосів
          </title>
        </head>
        <body>
          Правда про лосів.
          <ol>
            <li>
              Лось – тварина хитра
            </li>
            <li>
              .. і підступна
            </li>
          </ol>
        </body>
</html>
```

Рис. 5.22. **Приклад html-сторінки**

Кореневим елементом ієрархії є html. У нього є два нащадки. Перший – head, другий – body. Послідовність така: кожен вкладений тег є нащадком тегу, розташованого вище (рис. 5.23).

На цьому малюнку синім кольором позначені елементи-вузли, чорним – текстові елементи.

Дерево утворене за рахунок синіх елементів-вузлів – тегів HTML.

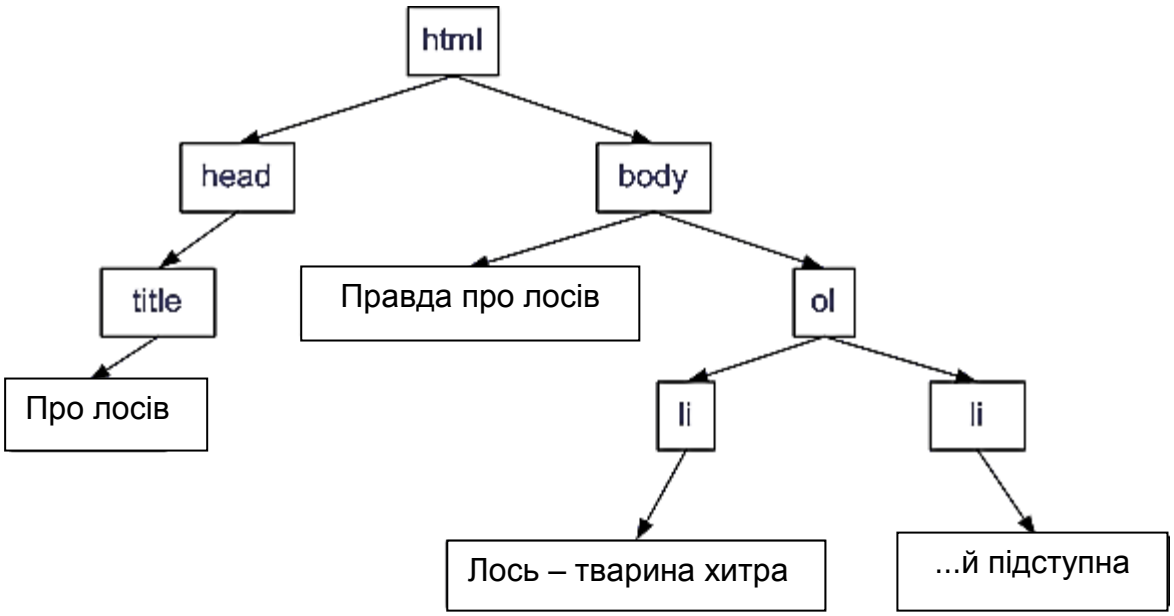


Рис. 5.23. **Ієрархічна модель DOM html-сторінки**

Так виглядає дерево, якщо зобразити його прямо на HTML-сторінці (рис. 5.24):

Рис. 5.24. **DOM на html-сторінці**

Об'єкт document

Об'єкт document є найважливішою властивістю об'єкта window (тобто повністю до нього потрібно звертатися як window.document). Усі елементи HTML-розмітки, присутні на веб-сторінці, – текст, абзаци,

гіперпосилання, малюнки, списки, таблиці, форми тощо – є властивостями об'єкта document. Можна сказати, що технологія DHTML (Dynamic HTML), тобто динамічна зміна вмісту веб-сторінки, полягає саме в роботі з властивостями, методами та подіями об'єкта document.

Уміння працювати з документом в моделі DOM є наріжним каменем у JavaScript-програмуванні.

У табл. 5.5 наведені властивості, методи і події об'єкта document.

Таблиця 5.5

Властивості, методи та події об'єкта document

Властивості	Методи	Події
URL	open()	Load
domain	close()	Unload
title		
lastModified	write()	Click
referrer	writeln()	DblClick
cookie		
	getSelection()	MouseDown
linkColor		MouseUp
alinkColor	getElementById()	
vlinkColor	getElementsByName()	KeyDown
	getElementsByTagName()	KeyUp
		KeyPress

Крім зазначених у цій таблиці, об'єкт document має властивості, які є колекціями (форм, малюнків, посилань тощо) та розглядалися в розділі клієнтських об'єктів і наведені в табл. 5.1.

Крім того, можна формувати необхідні колекції "на льоту" за допомогою зазначених вище методів. Так, document.**getElementsByTagName** ('P') є колекцією всіх HTML-елементів (точніше, відповідних до них об'єктів) виду <P>, тобто абзаців. Аналогічно, document.**getElementsByName** ('important') видасть колекцію (об'єктів) HTML-елементів будь-яких типів, яким був заданий атрибут NAME = "important". Нарешті, document.**getElementById** ('id5') видасть той HTML-елемент (якщо їх декілька, то перший), якому був заданий атрибут ID = "id5".

Наведені вище приклади також називають засобами пошуку елемента в DOM.

Корисним є приклад (рис. 5.25), в якому знаходяться (формуються колекції) всі нащадки елемента (<DIV>). За дане завдання відповідає метод **getElementsByTagName** з параметром ' ***** '.

```
<div id="d1">
        <ol id="ol1">
          <li id="li1">1</li>
          <li id="li2">2</li>
        </ol>
      </div>
      ...
      var div = document.getElementById('d1'); // знаходження елемента з ID==d1
      var elems = div.getElementsByTagName('*'); // Знаходження всіх нащадків елемента div
      // виведе послідовність: ol1, li1, li2
for(var i=0; i<elems.length; i++) alert(elems[i].id);
```

Рис. 5.25. **Пошук всіх нащадків елемента**

Метод document.**getElementsByName** працює тільки з тими елементами, для яких в специфікації явно передбачений атрибут name: це form, input, a, select, textarea і ряд інших, більш рідкісних.

Метод document.**getElementsByName** не працюватиме з іншими елементами типу div, p тощо.

Існують і інші способи пошуку у DOM: XPath, cssQuery й інші. Як правило, вони реалізуються JavaScript-бібліотеками для розширення стандартних можливостей браузерів.

Також є метод **getElementsByClassName** для пошуку елементів за класом, але він зовсім не працює в IE, тому в чистому вигляді ним не користуються.

Доступ до елементів

Будь-який доступ і зміни DOM беруть свій початок від об'єкта document.

Вершина дерева

document.documentElement

Самий верхній тег. У разі коректної HTML-сторінки, це буде <html>.
document.body. Тег <body>, якщо є в документі (зобов'язаний бути).

Типи DOM-елементів

У кожного елемента в DOM-моделі є тип. Його номер зберігається в атрибуті elem.nodeType. Всього в DOM розрізняють 12-ть типів елементів. Зазвичай використовується тільки один: Node.ELEMENT_NODE, номер якого дорівнює 1. Елементам цього типу відповідають HTML-теги.

Іноді корисний ще тип Node.TEXT_NODE, який дорівнює 3. Це текстові елементи.

Решта типів в JavaScript-програмуванні не використовуються.

Дочірні елементи

З вершини дерева можна піти далі вниз. Для цього кожен DOM-вузол містить масив усіх нащадків, окремо – посилання на першого й останнього нащадка та ще ряд корисних властивостей.

Усі дочірні елементи, включаючи текстові, знаходяться в масиві **childNodes**. Властивості **firstChild** і **lastChild** показують на перший і останній дочірні елементи та дорівнюють *null*, якщо нащадків немає.

Властивість **parentNode** вказує на батька. Наприклад, для <body> таким елементом є <html>.

Властивості **previousSibling** і **nextSibling** вказують на лівого та правого братів вузла.

Властивості елементів

У DOM-елементів є маса властивостей. Зазвичай використовується максимум третина з них. Деякі з них можна читати та встановлювати, інші – тільки читати.

Є ще й третій варіант, що зустрічається в IE, коли встановлювати властивість можна тільки під час створення елемента.

Слід розглянути ще деякі (не всі) властивості елементів, корисні при роботі з DOM.

tagName

Атрибут є у елементів-тегів і містить ім’я тега в верхньому регістрі, призначений тільки для читання.

style

Зміна індивідуальних стилів елементів.

В усіх HTML-елементів є властивість style. З її допомогою можна легко міняти стилі, які вказані для елемента в HTML-атрибуті style.

Наприклад, можна встановити element.style.width (рис. 5.26).

```
<input
        type="button"
        style="width: 300px"
        onclick="this.style.width =
                parseInt(this.style.width)-10+'px'"
        value="Скоротити на 10px"/>
```

Рис. 5.26. **Використання властивості style**

Є загальне правило заміни – якщо CSS-атрибут має дефіси, то для установки style потрібно замінити їх на верхній регістр букв.

Наприклад, для установки властивості **z-index** в 1 000, потрібно поставити:

element.style.zIndex = 1 000.

Усі властивості доступні для читання та запису, але читається/змінюється при цьому виключно значення **атрибута style** цього елемента. Тобто читати чинні на даний момент стилі елементу потрібно інакше.

Доступ до чинних стилів елементів

На жаль, для визначення діючих стилів, крім стандарту DOM2 існує ще реалізація, яка використовується в **I**nternet **E**xplorer. DOM2 визначає метод **getComputedStyle**() об'єкта **document.defaultView**, який повертає діючі стилі для комбінації елемента та псевдо-елемента (наприклад, 'hover' або 'active'):

```
var element = document.getElementById('testElement');
```

```
var computedStyle =
document.defaultView.getComputedStyle(element, null);
alert(computedStyle.display);
```

В **I**nternet **E**xplorer замість цього у кожного елемента є властивість **currentStyle**, аналогічна властивості style, але яка містить усі діючі стилі. Можливість враховувати псевдо-елементи в цьому варіанті відсутня. Тобто код, аналогічний попередньому, виглядав би в IE так:

```
var element = document.getElementById('testElement');
alert(element.currentStyle.display);
```

Обидва варіанти не складно об'єднати. Достатньо написати на початку скрипта:

```
if (typeof document.defaultView == 'undefined')
  document.defaultView = {};

if (typeof document.defaultView.getComputedStyle ==
'undefined')
{
  document.defaultView.getComputedStyle =
function(element,pseudoElement)
    {
      return element.currentStyle;
    }
}
```

Після цього можна скрізь користуватися функцією **document. defaultView.getComputedStyle**(). Якщо не використовувати псевдо-елементи, все буде працювати правильно в будь-якому браузері, в принципі підтримуючи доступ до діючих стилів.

innerHTML

Цю властивість містить весь HTML-код усередині вузла, і його можна змінювати.

Властивість innerHTML застосовується в основному для динамічної зміни змісту сторінки, наприклад:

```
document.getElementById('footer'). innerHTML =
'<h1>Bye!</h1> <p>See ya</p>';
```

Певно, innerHTML – одна з найбільш часто використовуваних властивостей DOM-елемента.

className

Ця властивість задає клас елементу. Вона повністю аналогічна html-атрибуту "class", наприклад:

elem**.**className **=** 'newclass'

Об'єкт Image

Найбільш видовищні ефекти у програмуванні на JavaScript досягаються під час роботи з графікою. При цьому в арсеналі програміста не так вже й багато інструментів: вбудовані в документ малюнки, можливість генерації об'єкта **Image**, комбінування малюнків з гіпертекстовими посиланнями та таблицями. Проте кількість різноманітних ефектів, які досягаються цими нехитрими засобами, вражає.

Програмування графіки в JavaScript спирається на об'єкт **Image**, який характеризується наступними властивостями, методами та подіями (табл. 5.6).

Незважаючи на значну кількість властивостей, їх абсолютну більшість можна тільки читати, але не змінювати. Про це свідчить, перш за все, відсутність методів. Але дві властивості все ж можна змінювати: src і lowSrc. Цього виявляється достатньо для безлічі ефектів з малюнками.

Таблиця 5.6

Характеристики об'єкта Image

Властивості	Методи	Події
name src lowSrc border height width hspace vspace complete	Не має	Abort Error Load

Усі об'єкти класу Image можна розділити на вбудовані та створені програмістом. Вбудовані об'єкти – це малюнки контейнерів IMG. Якщо ці малюнки найменувати, до них можна звертатися за іменем. Наприклад, якщо є малюнок (вважається, що він перший у документі):

<IMG NAME="picname" SRC="forest.gif"**>**,

то значення властивості document.images[0].name дорівнюватиме "picname", а до самого малюнка можна буде звертатися трьома способами:

```
document.images[0]
document.picname
document.images['picname']
```

Властивості src і lowSrc

Властивості src і lowSrc визначають URL-зображення, яке монтується всередину документа. При цьому lowSrc визначає тимчасове зображення, зазвичай невелике, яке відображується, поки завантажується основне зображення, чий URL вказується в атрибуті SRC контейнера IMG. Властивість src приймає значення атрибута SRC контейнера IMG. Програміст може змінювати значення і src, і lowSrc. У попередньому прикладі можна змінити значення src наступним чином:

```
document.picname.src='river.gif';
```

Як видно з цього прикладу, існує можливість модифікувати малюнок за рахунок зміни значення властивості src вбудованого об'єкта Image. Якщо дана сторінка переглядається вперше (тобто малюнок не закешировався браузером), то поступова зміна малюнка буде помітною (звичайно, за низької швидкості підключення до мережі інтернет; також це залежить від браузера, який може завантажити малюнок і тільки потім вивести його цілком на сторінку). Як прискорити цю зміну, розглядається в наступному розділі.

Зміна малюнка

Змінити малюнок можна, тільки присвоївши властивості *src* вбудованого об'єкта *Image* нове значення. Вище було показано, як це можна зробити в простому випадку. Очевидно, що повільне перезавантаження малюнка з сервера не дозволяє реалізувати швидке перегортання.

Рішення полягає в розведенні за часом підкачки малюнка та його відображення. Для початку створюється зображення, до якого прив'язуються обробники подій onMouseOver і onMouseOut. Після наведення покажчика миші на кожен з малюнків він замінюється іншим (кольоровим), а при відведенні мишки малюнок знов стає чорно-білим (рис. 5.27).

```
<IMG NAME=m0 src="images/mapb000.gif" border=0
 onMouseOver="document.m0.src=color[0].src;"
  onMouseOut="document.m0.src= mono[0].src;">
```

Рис. 5.27. **Установка малюнка з обробниками**

Однак головне не в тому, що малюнки заміщуються, а в тому, з якою швидкістю це відбувається. Для досягнення потрібного результату на початку сторінки створюються масиви малюнків, в які перед відображенням перекачується вся графіка. З цією метою використовують конструктор об'єкта Image (рис. 5.28).

```
mono = new Array(32);
for(i=0;i<32;i++)
{
   mono[i]  = new Image();
   s = (i<10)? "0" : "";
   mono[i].src  = "images/mapb0" +s+i+ ".gif";
}
```

Рис. 5.28. **Створення масиву малюнків**

Саме в той момент, коли властивості, наприклад, *mono[25].src* присвоюється значення "*images/mapb025.gif*", відбувається скачування цього малюнка з сайту на комп'ютер користувача. Якби замість об'єктів класу **Image** масив був складений з рядків виду "*images/mapb000.gif*" тощо, то підвантаження графіки не відбулося б, і кожен раз, коли користувач наводив би на чергове зображення, браузеру доводилось би скачувати нову картинку. Цей скрипт був поміщений у контейнер <HEAD>, тим самим гарантувавши, що до моменту, коли користувач почне працювати зі сторінкою, всі необхідні для роботи малюнки вже будуть викачані, і в процесі водіння мишкою по малюнках жодної затримки показу чергового зображення спостерігатися не буде.

Також варто звернути увагу на наступний трюк, використаний в скрипті. Припустимо, що необхідно написати 32 рядка:

```
document.m0.src = mono[0].src;
document.m1.src = mono[1].src;
...
document.m31.src = mono[31].src;
```

Необхідно уникати написання такого громіздкого коду. З правою частиною операторів присвоювання впоратися легко: достатньо задати

цикл по і від 0 до 31 і писати mono [i].src. А от з лівою частиною не все так просто. На допомогу приходить функція **eval** ("вираз"), яка сприймає переданий їй вираз як програму JavaScript і виконує всі оператори, що містяться в ній. З її допомогою вирішити проблему легко – складається потрібний рядок, який віддається на виконання:

```
for(i=0;i<32;i++)
  eval("document.m" +i+ ".src = mono[" +i+ "].src;");
```

Мультиплікація

Природним продовженням ідеї заміщення значення атрибута SRC у контейнері IMG є мультиплікація, тобто послідовна зміна значення цього атрибута в часі. Для реалізації мультиплікації використовують метод setTimeout() об'єкта window.

Існує два способи запуску мультиплікації: по закінченні завантаження сторінки (**onLoad**) і при діях користувача (**onClick**, **onChange** тощо). Найбільш популярний – перший, а саме – використання **onLoad**() і **setTimeout**().

Обробник події **onLoad**

Подія Load наступає в момент закінчення завантаження документа браузером. Оброблювач даної події (onLoad) вказується в контейнері BODY:

```
<BODY onLoad="програма JavaScript">
```

На рис. 5.29 наведений приклад, в якому під час завантаження документа починає виконуватися нескінченний цикл зміни малюнка:

```
</HEAD>
 <SCRIPT>
   var i=0;
   function movie()
   {
     document.i.src='images/crou00'+i+'.gif';
     i++; if(i>4) i=0; setTimeout('movie();',500);
   }
 </SCRIPT>
</HEAD>
<BODY onLoad="movie();">
  <IMG NAME="i">
</BODY>
```

Рис. 5.29. **Нескінченна мультиплікація**

На перший погляд, це просто рекурсія. Але насправді все дещо складніше. JavaScript розроблявся для багатопоточних операційних систем, тому правильніше буде уявляти собі виконання скриптів наступним чином:

скрипт movie() отримує управління від обробника події onLoad;

замінює малюнок;

породжує новий скрипт movie() і відкладає його виконання на 500 мілісекунд;

поточний скрипт movie() знищується JavaScript-інтерпретатором.

Після закінчення терміну затримки виконання безупинно повторюється.

Запуск і зупинка мультиплікації

Постійна мультиплікація може бути досягнута й іншими засобами, наприклад, багатокадрова – графічними файлами. Однак рух на сторінці – не завжди благо. Часто виникає бажання реалізувати запуск і зупинити рух на вимогу користувача. Для цього можна модифікувати приклад з нескінченною мультиплікацією (результат поданий на рис. 5.30).

```
<HEAD>
        <SCRIPT>
          var i=0, flag=true;
          function movie()
          {
           if(flag)
           {
            document.i.src='images/crou00'+i+'.gif';
            i++; if(i>4) i=0;
           }
           setTimeout('movie();',500);
          }
        </SCRIPT>
      </HEAD>
      <BODY onLoad="movie();">
        <FORM>
        <INPUT TYPE="button" VALUE="Start/Stop"
              onClick="flag = !flag;">
        </FORM>
        <IMG NAME="i">
</BODY>
```

Рис. 5.30. **Зупинка/запуск мультиплікації**
(потік генерується постійно)

Введенням булевського flag зображення на сторінці можна змінити тільки, якщо він приймє значення true. Натискання на кнопку змінює значення "прапора" на його заперечення.

Слід звернути увагу: коли flag має значення false, треба просто обходити зміну малюнка, не припиняючи породження потоку. Якщо помістити setTimeout() всередину конструкції if(), то після натискання на кнопку потік перестав би породжуватися, і запустити мультиплікацію заново не можна. Проте постійне генерування потоку – це певна розтрата ресурсів (пам'яті, процесора). Чи можна зробити це оптимальніше? Можна. Цей спосіб заснований на застосуванні методу clearTimeout() (рис. 5.31).

```
<HEAD>
        <SCRIPT>
          var i=0, flag=true, m=null;
          function movie()
          {
           if(flag)
           {
             document.i.src='images/crou00'+i+'.gif';
             i++;
       if(i>4) i=0;
           }
           m = setTimeout('movie();',500);
          }
        </SCRIPT>
      </HEAD>
      <BODY onLoad="movie();">
        <IMG NAME=i>
        <FORM>
          <INPUT TYPE=button VALUE="Start/Stop"
            onClick="flag = !flag;
            if(flag) movie();
            else clearTimeout(m);">
        </FORM>
</BODY>
```

Рис. 5.31. **Зупинка/запуск мультиплікації**
(при зупинці потоку)

Достатньо ввести ідентифікатор потоку m і зберігати в ньому посилання на потік викликом setTimeout(). Тоді за потребою (натисканням

кнопки) можна скасувати заплановане виконання movie() (яке відбулося б через 500 мілісекунд), викликавши метод clearTimeout(m).

Оптимізація відображення

Під час програмування графіки слід враховувати безліч чинників, які впливають на швидкість відображення сторінки та швидкість зміни графічних образів. При цьому звичайна дилема оптимізації програм – швидкість або розмір займаної пам'яті – вирішується тільки на користь збільшення швидкості. Розмір пам'яті у програмуванні на JavaScript на увагу не береться.

З усіх способів оптимізації відображення малюнків слід зупинитися тільки на декількох:

оптимізація відображення під час завантаження;

оптимізація відображення за рахунок попереднього завантаження;

оптимізація відображення за рахунок нарізки зображення.

Якщо перші дві позиції належать як до відображення статичних малюнків, так і до мультиплікації, то третій пункт характерний головним чином для мультиплікації.

Оптимізація під час завантаження зображень

В усіх посібниках з розроблення HTML-сторінок зазначається, що під час використання контейнера IMG в тілі HTML-сторінки слід вказувати атрибути WIDTH і HEIGHT. Це продиктовано порядком завантаження компонентів сторінки з сервера й алгоритмом роботи HTML-парсера. Першим завантажується текст розмітки. Після цього парсер розбирає текст і починає завантаження додаткових компонентів, у тому числі графіки. При цьому завантаження малюнків, залежно від типу HTTP-протоколу, може йти послідовно або паралельно.

Також паралельно із завантаженням парсер продовжує свою роботу. Якщо для малюнків задані параметри ширини та висоти, то можна відформатувати текст і відобразити його у вікні браузера. Доки ці параметри не визначені, відображення тексту не відбувається. У сучасних браузерах текст зазвичай відображується одразу, залишаючи для малюнка "невелике" місце; по закінченні завантаження малюнка (або коли стають відомими його розміри) сторінка переформатовується "на льоту".

Таким чином, вказівка на висоту та ширину малюнків дозволить відобразити документ раніше, ніж малюнки будуть отримані з сервера.

Це дає користувачеві можливість читати документ або задіяти його гіпертекстові посилання до моменту повного завантаження документа.

JavaScript вказівка на розміри малюнка задає початкові параметри вікна відображення графіки усередині документа. Це дозволяє скористатися невеликим прозорим чином, для того щоб замінити його повноцінним малюнком. Ідея полягає в передаванні невеликого об'єкта для заміщення його, за потребою, великим об'єктом.

Попереднє завантаження зображень

Заміна одного образу іншим часто буває виправдана тільки тоді, коли це відбувається досить швидко. Якщо перезавантаження триває довго, то ефект втрачається. Для швидкої підміни використовують можливість попереднього завантаження документа в спеціально створений об'єкт класу Image.

Реальний ефект можна відчути тільки після відключення кешування сторінок на боці клієнта (браузера). Кешування часто використовують для прискорення роботи зі сторінками Web-вузла. Як правило, завантаження першої сторінки – досить тривалий процес. Найголовніше, щоб користувач у цей момент був готовий зачекати. Тому крім графіки, необхідної тільки на першій сторінці, йому можна передати і графіку, яка на ній не відображується. Зате під час переходу до інших сторінок вузла вона буде відображуватися без затримки на передачу з сервера.

Описаний вище прийом неоднозначний. Його виправдовує тільки те, що якщо користувач нетерплячий, то він взагалі відключить передачу графіки.

Нарізка зображень

Нарізка малюнків застосовується досить часто. Вона дозволяє досягати ефекту, часткової зміни відображуваного малюнка, який найчастіше застосовується при створенні меню.

Крім подібного ефекту нарізка дозволяє реалізувати мультиплікацію на великих малюнках. При цьому змінюється не весь образ, а тільки окремі його частини.

Графіка та таблиці

Одним з найбільш популярних прийомів дизайну сторінок веб-вузла є техніка нарізки малюнків на складові частини. Можна виділити наступні

способи застосування цієї техніки для організації навігаційних компонентів сторінки:

горизонтальні та вертикальні меню;

вкладені меню;

навігаційні графічні блоки;

горизонтальне меню.

Головною проблемою у використанні нарізаної графіки є її захист від контекстного форматування сторінки HTML-парсером. Справа в тому, що він автоматично переносить елементи розмітки на новий рядок, якщо вони не поміщаються на одному. Складові частини нарізаного малюнка повинні бути розташовані на екрані певним чином, але просте їх перерахування в ряд не дає бажаного ефекту.

Проблема вирішується застосуванням "захисту" від перенесення на наступну сходинку – до контейнера <PRE> (рис. 5.32).

```
<PRE>
        <IMG SRC="horis1.gif"><IMG
            SRC="horis2.gif"><IMG
            SRC="horis3.gif"><IMG
            SRC="horis4.gif">
</PRE>
```

Рис. 5.32. **Горизонтальне меню**
(малюнки захищені від переносу)

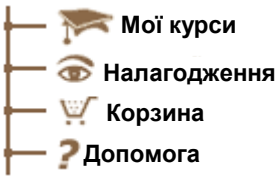
Вертикальне меню

У той же спосіб можна реалізувати вертикальне меню. Під час перегляду в деяких браузерах суцільна вертикальна лінія не створюється, тому що висота рядка не дорівнює висоті малюнка. Підігнати ці параметри практично неможливо. Кожен користувач налаштовує браузер за власним бажанням.

Рішення полягає у використанні (замість контейнера <PRE>) таблиці <TABLE> (рис. 5.33) з нульовими межами між осередками та нульовими полями осередків.

У даному випадку всі малюнки можна зшити без пропусків і тим самим досягти безперервності навігаційного дерева.

```
<STYLE type="text/css">
        #menu_tab{
          border-collapse:0px; border-spacing:0px;
        }
        #menu_tab, #menu_tab td{ padding:0px; }
        #menu_tab, img.img_vert, img.img_vert_item{ border:0px;
}
        #menu_tab img.img_vert{
          width:27px; height:21px;
        }
        #menu_tab img.img_vert_item{
          width:103px; height:21px;
        }</STYLE>
      <TABLE id="menu_tab">
        <TR>
          <TD> <IMG SRC="vert.gif" class="img_vert"> </TD>
          <TD>
            <A HREF="courses.htm">
              <IMG SRC="vert1.gif" class="img_vert_item">
            </A>
          </TD>
        </TR>
        <!-- далі аналогічно для vert2, vert3, vert4.gif -->
</TABLE>
```



А Б

Рис. 5.33. **Вертикальне меню**

Виділення обраного пункту меню

Вищевикладене стосується питань побудови однорівневих меню. Тому в даному розділі слід привести реальні приклади таких меню. Графічне меню зручне тим, що автор може завжди досить точно розташувати його компоненти на екрані. Це, в свою чергу, дозволяє й інші елементи сторінки точніше розташувати щодо елементів меню.

Оскільки пункти меню прилягають один до одного без просвітів, то пункт меню, активний на даний момент, буде позначений стрілочкою (рис. 5.34).

```
<STYLE type="text/css">
        #menu_tab {
      border-collapse:0px;
      border-spacing:0px;
      }
        #menu_tab, #menu_tab td{ padding:0px; }
        #menu_tab, img.bullet_img, img.img_item { border:0px; }
        #menu_tab img.bullet_img { width:15px; height:8px; }
        #menu_tab img.img_item { height:42px; }
        tr.bullet_tr { text-align:center; }
      </STYLE>
      ...
      <TABLE id="menu_tab">
       <TR class="bullet_tr">
        <TD><IMG src="empty.gif" name="e1" class="bullet_img"></TD>
        <!-- аналогично для e2, e3, e4 -->
       </TR>
       <TR>
        <TD>
          <A HREF="javascript:void(0);"
             onmouseover="document.e1.src='arrowdw.gif';"
             onmouseout="document.e1.src='empty.gif';">
               <IMG src="horis1.gif" class="img_item">
          </A>
        </TD>
        <!-- аналогічно для e2, e3, e4 -->
       </TR>
</TABLE>
```

Рис. 5.34. **Горизонтальне меню**
(з індикатором)

Стрілочка біжить саме над тим елементом, на який вказує миша.

Проте варто зауважити, що застосування атрибутів ALT і TITLE у контейнера IMG та їх дублювання в рядку статусу є більш інформативним, ніж додавання нового графічного елементу. Однак відображується зміст TITLE з деякою затримкою (рис. 5.35).

```
<STYLE type="text/css">
        a img { border:none; }
    </STYLE>
    ...
    <PRE>
    <A HREF="courses.html" onmouseover="window.status=' Мої курси';return
true;" onmouseout="window.status='';">
    <IMG src="horis1.gif" class="img_item" ALT="Мої курси" TITLE="Мої курси">
    </A>
    <A HREF="setting.html" onmouseover="window.status='Налагодження';return
true;" onmouseout="window.status='';">
    <IMG src="horis2.gif" class="img_item" ALT="Налагодження"
TITLE="Налагодження">
    </A>
    <A HREF="baskets.html" onmouseover="window.status='Корзина';return true;"
onmouseout="window.status='';">
    <IMG src="horis3.gif" class="img_item" ALT="Корзина" TITLE="Корзина">
    </A>
    <A HREF="thehelp.html"
    onmouseover="window.status='Допомога';return true;"
    onmouseout="window.status='';">
    <IMG src="horis4.gif" class="img_item" ALT="Допомога" TITLE="Допомога">
    </A>
</PRE>
```

Рис. 5.35. **Горизонтальне меню**
(з атрибутами ALT, TITLE і полем статусу)

Приклад вирішення завдання

Як приклад розглядаються модулі (блоки), які складаються з заголовка й основного вмісту. Після клацання по заголовку модуля блок основного вмісту згортається, після повторного клацання – розгортається.

Для вирішення цього завдання він розібивається на складові: розмітка модуля за допомогою HTML (рис. 5.36), стилізація за допомогою таблиць стилів (рис. 5.37) і власне реалізація інтерактивності за допомогою JavaScript (рис. 5.38).

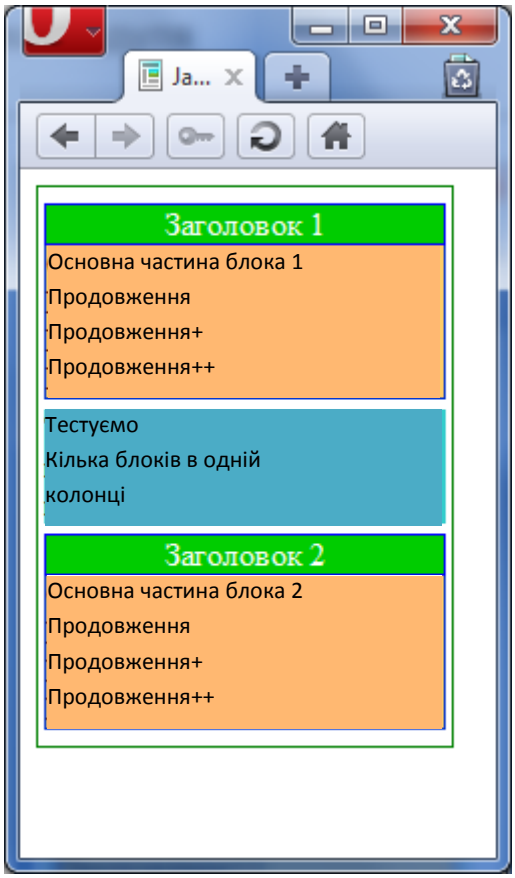
```
<div class="left_column">
        <div id="block_01">
          <div class="head">Заголовок 1</div>
          <div class="main">Основна частина блока 1<br
/>Продовження<br />
Продовження +<br /> Продовження ++</div>
        </div>
        <div class="other">Тестируємо<br />кілька блоків в одній
колонці</div>
        <div id="block_02">
          <div class="head">Заголовок 2</div>
          <div class="main">Основна частина блока 2<br />
Продовження <br />
Продовження +<br /> Продовження ++</div>
        </div>
</div>
```

Рис. 5.36. **Розмітка модуля**

Оформлення за допомогою CSS НЕ не впливає на реалізацію інтерактивності модулів, воно потрібно тільки для подання.

Для створення універсальної функції її потрібно зробити параметризованою і як параметр передати покажчик (this) на поточний елемент DOM. Дана функція викликається в обробнику події click заголовка модуля (рис. 5.38). У самій функції для згортання основної частини модуля (... </ div>) можна змінювати його css-властивість **display** (none – приховати, block – відобразити). А для отримання доступу до основного блоку потрібно провести пошук усіх елементів з тегом " " у батьківському елементі ( ) за допомогою методу "getElementsByTagName ('div')" (рис. 5.38) і в отриманому масиві елементів знайти елемент з ім’ям класу основного блоку.

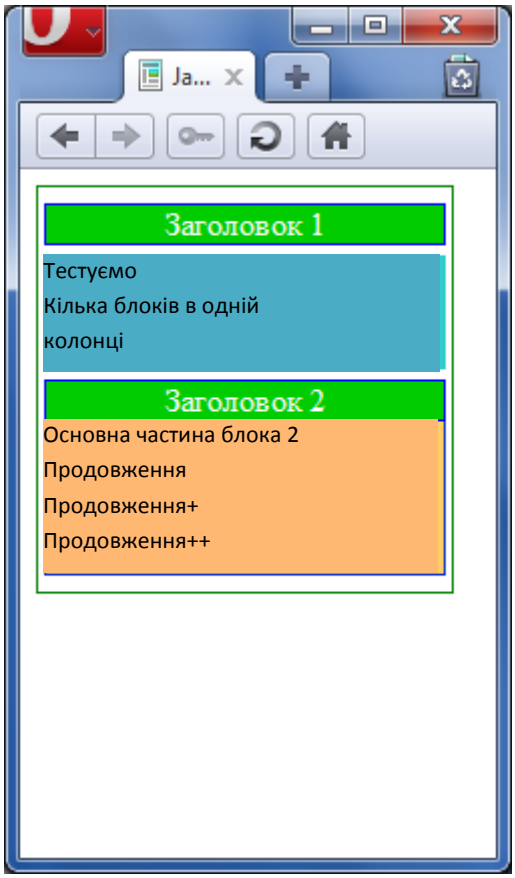
```
<STYLE type="text/css">
        /* Ліва колонка каркаса сторінки */
        div.left_column{
            width:200px;
            border:1px solid green;
            padding:3px;
        }
        /* Заголовок блоків */
        div.head{
            text-align:center;
            background-color:#0C0;
            color:#FFF;
            border:1px solid blue;
            cursor:pointer;
        }
        /* Основна частина блоків */
        div.main{
            background-color:#FC6;
            color:#000;
            border-width: 0px 1px 1px 1px;
            border-style:solid;
            border-color:#03C;
        }
        div.other{
            background-color:#3CC;
        }
        /* Відсупи зверху та знизу від кожного
блока всередині лівої колонки */
        div.left_column> div{
            margin:5px 0px;
        }
</STYLE>
```



А Б

Рис. 5.37. **Оформлення модуля за допомогою CSS**

```
<script type="text/javascript">
        function Collaps_Exp(header)
        {
            var main_block =
header.parentNode.getElementsByTagName('div');
            var main;
            for(var i=0; i<main_block.length; i++)
            {
                if(main_block[i].className=='main')
                {main = main_block[i]}
            }
            if(main.style.display=='none')
            {
                main.style.display='block';
            }
            else
            {
                main.style.display='none';
            }
        }
    </script>
    <!--... -->
    <div class="head"
onclick="Collaps_Exp(this);">
Заголовок 1</div>
    <!--... -->
    <div class="head"
onclick="Collaps_Exp(this);">
Заголовок 2</div>
```



А Б

Рис. 5.38. **Реалізація JavaScript-функції згортання/розгортання модуля**

5.4. Завдання до лабораторної роботи

1. У сайт, створений в рамках лабораторної роботи №4, додати функціональність, наведену в модульній сітці відповідно до варіанту.

2. У блоці "Copyright" під час завантаження сторінки вставити поточну дату (у варіанті 7 – в контактні реквізити).

3. Після наведення миші на область з датою в ній починає відображатися поточний час (год; хв; с), який оновлюється кожну

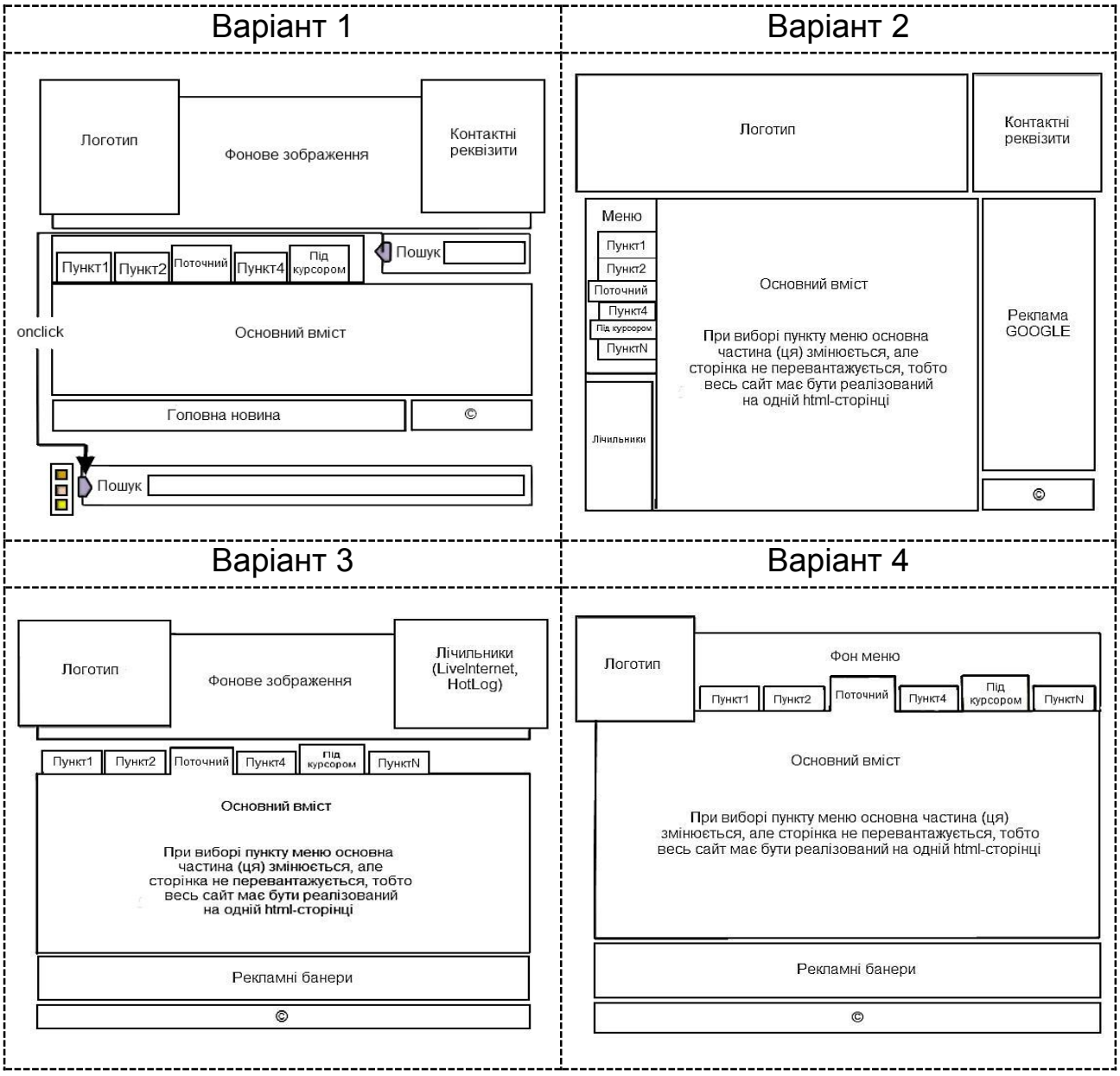
секунду. Після відведення курсора миші з даної області в ній знову відображується поточна дата.

4. Логотип або інше (більш слушне, на ваш погляд) зображення реалізувати декількома зображеннями. Після наведення курсору миші на логотип деякі зображення, з яких він складається, повинні замінюватися на більш яскраві (більш цікаві рішення вітаються).

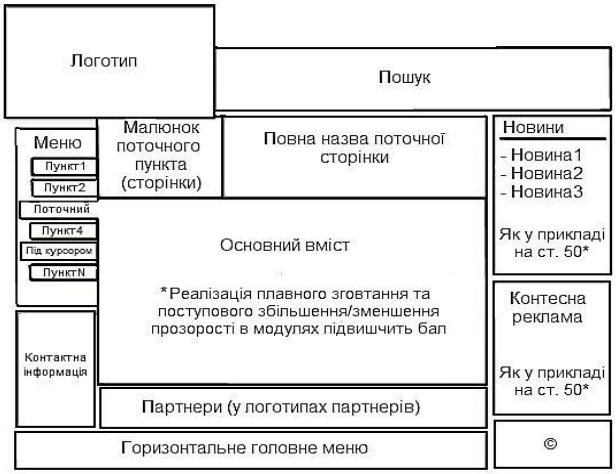
5. Завантаження зображень повинно бути оптимізоване.

6. Клацанням по логотипу повинно створюватися вікно 800x600px, в яке завантажиться головна сторінка сайта JavaScript.ru.

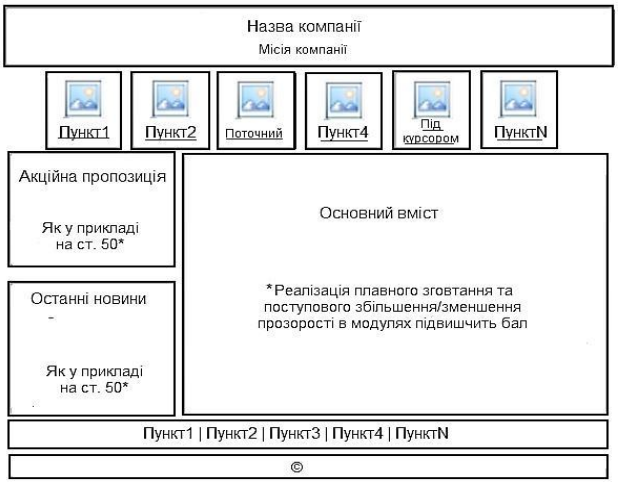
Варіанти завдання



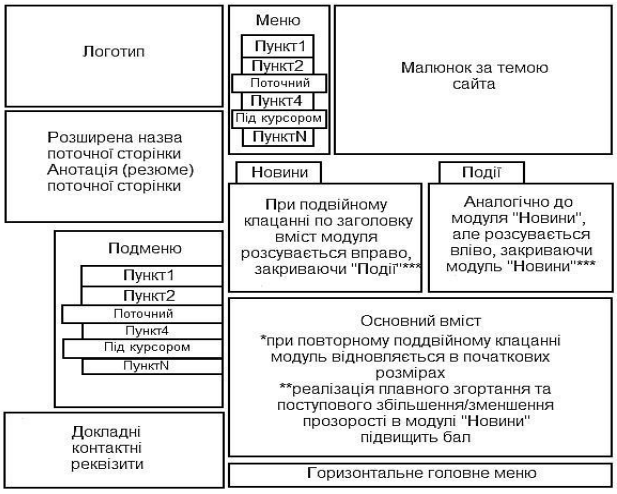
Варіант 5



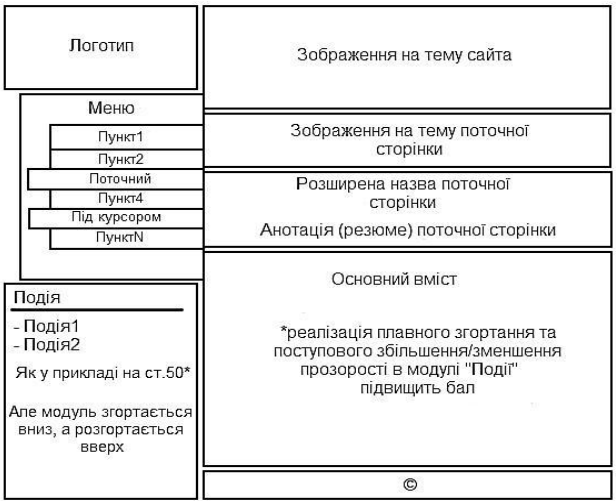
Варіант 6



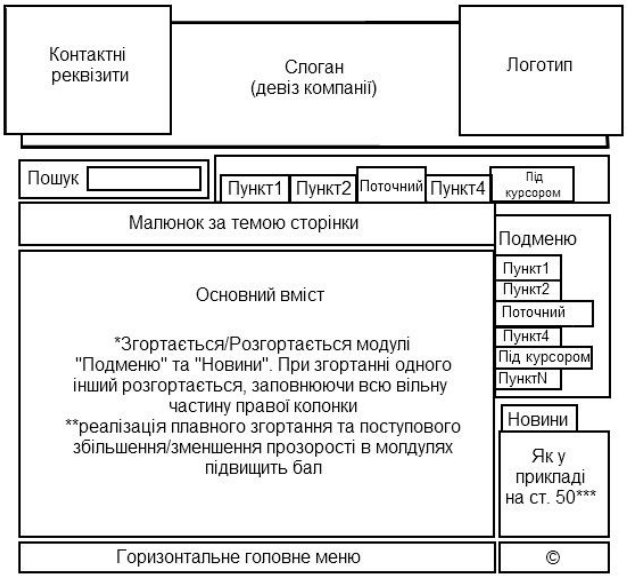
Варіант 7



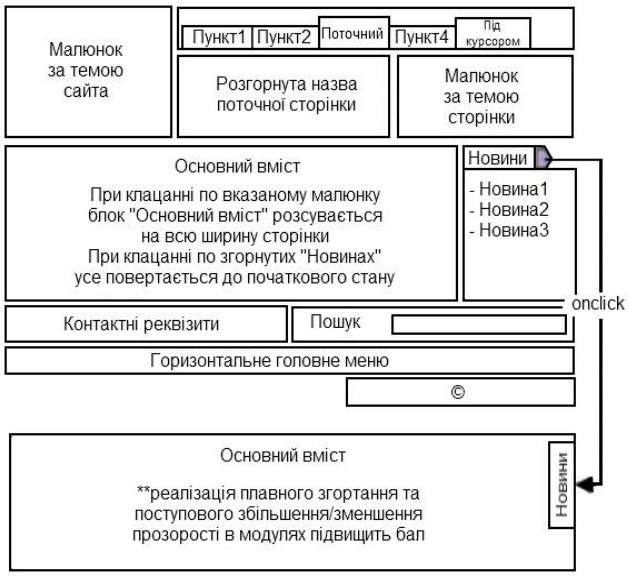
Варіант 8



Варіант 9



Варіант 10



5.5. Контрольні запитання

1. Що таке DHTML?
2. Що таке DOM?
3. Назвіть основну структуру DOM, опишіть її основні об'єкти.
4. Які типи об'єктів існують в JavaScript?
5. Назвіть види використання ключового слова this.
6. Чим відрізняється спосіб зміни індивідуальних стилів елемента від способу доступу до чинних стилів елемента?
7. Чому часто використовується властивість innerHTML?
8. Як знайти елемент за його ідентифікатором?
9. Як сформувати колекцію елементів одного тегу?
10. Як сформувати колекцію елементів одного класу?

Лабораторна робота № 6
Розроблення динамічних веб-сторінок за допомогою JavaScript-бібліотеки jQuery

6.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення можливостей бібліотек JavaScript, зокрема jQuery.

6.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Під час підготовки до лабораторної роботи необхідно вивчити основні можливості бібліотек JavaScript, зокрема jQuery.

6.3. Загальні положення лабораторної роботи

Бібліотека JavaScript – збірка класів та / або функцій на мові JavaScript [3; 6; 8; 12].

Зі збільшенням популярності JavaScript простота створення динамічних елементів користувальницького інтерфейсу почала відігравати ключову роль у веб-розробленнях. Цим обумовлений лавиноподібний характер появи різноманітних бібліотек JavaScript – таких, як Ext і Dojo. З іншого боку, одним з наслідків "війни" браузерів є відмінність у реалізації об'єктної моделі документа, і це спричинило необхідність у додаткових зусиллях для реалізації коректної роботи різних браузерів. Дана обставина зумовила появу бібліотек JavaScript, які пропонують крос-браузерні інтерфейси до методів DOM, таких, як: Prototype, Yahoo! UI, Mootools або jQuery.

Як предмет докладного розгляду в даній лабораторній роботі обрана бібліотека jQuery.

jQuery – бібліотека JavaScript, яка фокусується на взаємодії JavaScript та HTML. Бібліотека jQuery допомагає легко отримувати доступ до будь-якого елемента DOM, звертатися до атрибутів і вмісту елементів DOM, маніпулювати ними. Також бібліотека jQuery надає зручний API по роботі з Ajax [12; 15].

Так само, як CSS відокремлює візуалізацію від структури HTML, JQuery відділяє поведінку від структури HTML. Наприклад, замість прямої вказівки на обробник події натискання кнопки управління передається JQuery, яке ідентифікує кнопку і потім перетворює його в

обробник події кліка. Таке поділення поведінки та структури називається принципом "ненав'язливого" JavaScript.

Бібліотека jQuery містить функціонал, корисний для максимально широкого кола завдань. Проте розробниками бібліотеки не ставилося завдання суміщення в jQuery універсальних функцій, оскільки це призвело б до громіздкого коду, значна частина якого буде не затребувана. Тому була реалізована архітектура компактного універсального ядра бібліотеки та плагінів. Це дозволяє зібрати для ресурсу саме той JavaScript-функціонал, який був би затребуваний.

Використання

jQuery, як правило, включається в веб-сторінку як один зовнішній JavaScript-файл (рис. 6.1).

```
<head>
<script type="text/javascript"
src="шлях/до/jQuery.js">
</script>
</head>
```

Рис. 6.1. **Підключення бібліотеки jQuery**

Уся робота з jQuery ведеться за допомогою функції $. Якщо на сайті застосовуються інші JavaScript-бібліотеки, де $ може використовуватися для їх потреб, то можна використовувати її синонім – jQuery. Другий спосіб вважається більш правильним, а щоб код не виходив занадто громіздким, можна писати його наступним чином (рис. 6.2):

```
jQuery(function($) {
  // Тут код скрипта, де в $ буде jQuery
})
```

Рис. 6.2. **Використання jQuery**

Роботу з jQuery можна розділити на 2 типи:

отримання jQuery-об'єкта за допомогою функції **$ ()**. Наприклад, передавши до неї CSS-селектор, можна отримати jQuery-об'єкт усіх

елементів HTML, які потрапляють під критерій, і далі працювати з ними за допомогою різних методів jQuery-об'єкта;

виклик глобальних методів у об'єкта **$,** наприклад зручних ітераторів по масиву.

Основні способи роботи з бібліотекою jQuery подані на рис. 6.3.

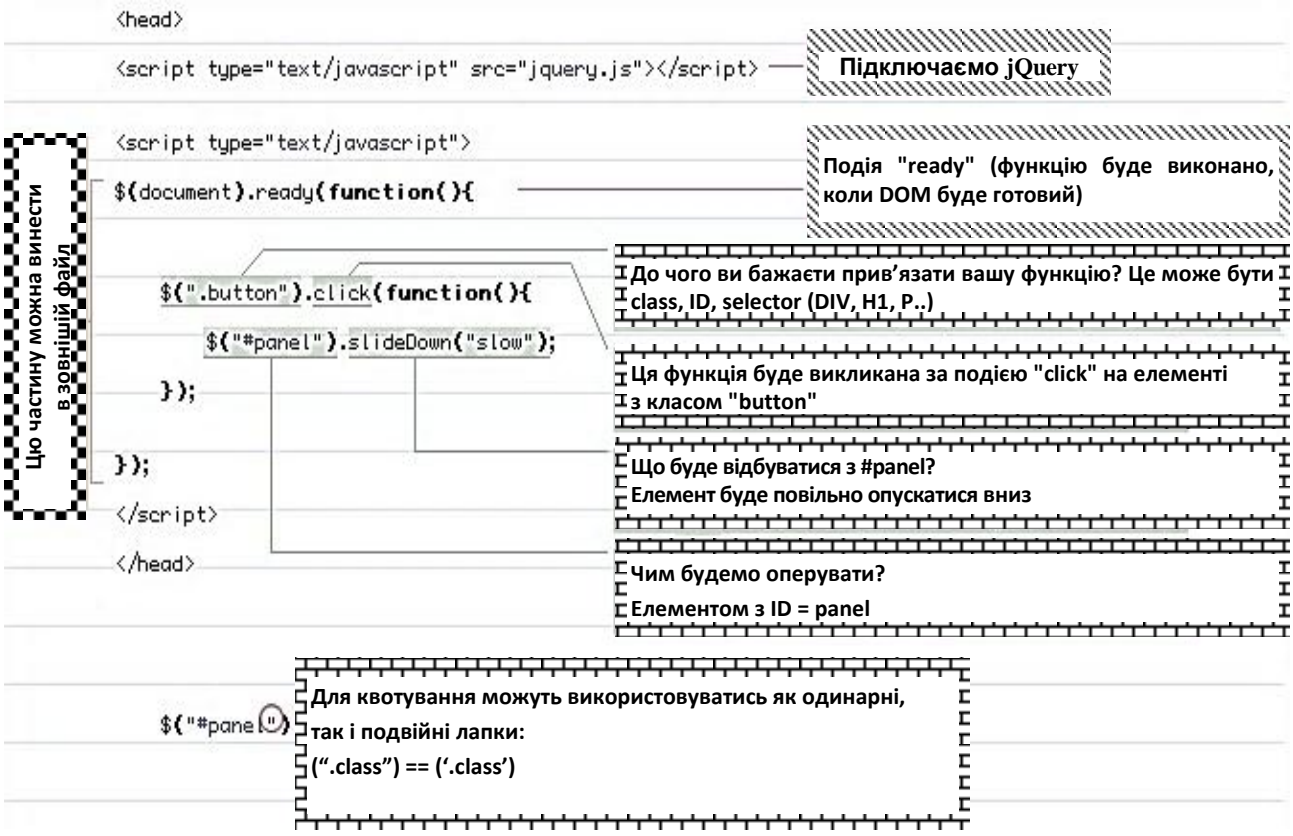


Рис. 6.3. **Основні способи роботи з бібліотекою jQuery**

Як отримати елемент за допомогою jQuery?

Для більш докладного розгляду селекторів в jQuery користуються типовим макетом HTML-сторінки (рис. 6.4).

Для того щоб розуміти, як працює селектор, необхідні базові знання CSS, бо саме від принципів CSS відштовхується селектор jQuery:

$ ("# Header") – отримати елемент з id = "header";

$ ("H3") – отримати всі <h3> елементи;

$ ("Div # content .photo") – отримати всі елементи з класом = "photo", які знаходяться в елементі div з id = "content";

$ ("Ul li") – отримати всі <li> елементи зі списку <ul>;

$ ("Ul li: first") – отримати тільки перший елемент <li> зі списку <ul>.

```
<div id="header">
   <h1><a href="/" title="homepage">Title</a></h1>
   <h2>Sub-title <span>small description</span></h2>
</div>
<div id="wrapper">
  <div id="content">
    <div class="post">
      <h3>Post Title</h3>
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
         <span>Image Title</span>
         <img src="/image1.jpg" alt="Image Alt Text"/>
         <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
         <span class="inner-banner">Banner Text</span>
         <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
    </div>
    <span id="banner"><img src="/banner1.jpg" alt="Big Banner"/></span>
    <div class="post">
      <h3>Post Title</h3>
         <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
         <span>Image Title</span>
         <img src="/image2.jpg" alt="Image Alt Text"/>
         <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
         <span class="inner-banner">Banner Text</span>
         <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
        </div>
    </div>
</div>
<div id="sidebar">
    <ul>
        <li><a href="/item0.html">Menu Item 0</a></li>
        <li><a href="/item1.html">Menu Item 1</a></li>
        <li><a href="/item2.html">Menu Item 2</a></li>
        <li><a href="/item3.html">Menu Item 3</a></li>
    </ul>
</div>
<div id="footer">
    Copyright &copy; 2010
</div>
```

Рис. 6.4. **Макет HTML-сторінки**

Вибір елементів з Id або ClassName аналогічний використовуваному в CSS (рис. 6.5).

```
$('#sidebar');    // вибір елемента з id = sidebar
$('.post');       // вибір елемента з class = post
$('div#sidebar'); // вибір елемента div з id = sidebar
$('div.post');    // вибір елемента div з class = post
```

Рис. 6.5. **Вибір елементів по Id або ClassName**

Переміщення по ієрархії об'єктів у DOM'і

Простий вибір нащадків:

```
$('div span'); // вибір всіх span-елементів в елементах div
```

Аналогічний результат можна отримати, використовуючи наступну конструкцію:

```
$('div').find('span'); // вибір всіх span–елементів в елементах div
```

Вибір тільки безпосередніх нащадків:

```
$('div > span'); // вибір всіх span- елементів, які є прямими нащадками в div
```

Так само селектори можна групувати:

```
$('div, span'); // вибір всіх div- і span-елементів
```

Пошук по сусідах наведено на рис. 6.6.

```
$('span + img');          // вибір всіх img – елементів, перед якими йдуть span елементи
$('span ~ img');          // вибір всіх img – елементів після першого елемента span
$('#banner').prev();      // вибір попереднього елемента від знайденого
```

Рис. 6.6. **Пошук по сусідах**

Вибір всіх елементів, предків та нащадків наведено на рис. 6.7.

```
$('*');                   // вибір усіх елементів
$('p > *');               // вибір усіх нащадків елементів p
$('p').children();        // --
$('p').parent();          // вибір усіх прямих предків елементів p
$('* > p');               /* вибір усіх предків елементів p (скоріше за все, вам не знадобиться) */
$('p').parents();         // --
$('p').parents('div'); /* вибір усіх предків елемента p, які є div (parents приймає як параметр селектор) */
```

Рис. 6.7. **Вибір всіх елементів, предків та нащадків**

Фільтри

Фільтрів в jQuery реалізовано досить багато, і використовувати їх досить просто (рис. 6.8).

```
$('div:first');          // вибираємо перший div у DOM'і
$('div:last');           // вибираємо останній div у DOM'і
$('div:not(.red)');      // вибираємо div'и, у яких нема класу red
$('div:even');           // вибираємо парні div'и
$('div:odd');            // вибираємо непарні div'и
$('div:eq(N)');          // вибираємо div, що йде під номером N у DOM'і
$('div:gt(N)');          // вибираємо div'и, індекс яких більше за N у DOM'і
$('div:lt(N)');          // вибираємо div'и, індекс яких менший за N у DOM'і
$(':header');            // вибір заголовка h1, h2, h3 тощо
$('div:animated');       // вибір елементів з активною анімацією
```

Рис. 6.8. **Фільтри в jQuery**

Фільтри за контентом і видимістю наведені на рис. 6.9.

```
$('div:contains(text)');      // вибираємо div'и, які містять текст
$('div:empty');               // вибираємо порожні div'и
$('div:has(p)');              // вибираємо div'и, які містять p
$('div.red').filter('.bold')  // вибираємо div'и, які містять клас red і
клас bold
$('div:hidden');              // вибираємо приховані div'и
$('div:visible');             // вибираємо видимі div'и
```

Рис. 6.9. **Фільтри за контентом та видимістю**

У jQuery існують також фільтри за атрибутами (рис. 6.10).

```
$("div[id]");          // вибір всіх div з атрибутом id
$("div[title='my']");  // вибір всіх div з атрибутом title=my
$("div[title!='my']"); // вибір всіх div з атрибутом title, не рівним my
$("div[title^='my']"); /* вибір всіх div з атрибутом title, що
                       починаються з my <div title="myCat">,
                       <div title="myCoffee">, <div title="my..."> */
$("div[title$='my']"); /* вибір всіх div з атрибутом title, які закінчуються
                       на my <div title="itsmy">,<div title="somy">,
                       <div title="...my"> */
$("div[title*='my']"); /* вибір всіх div з атрибутом title, що містить my
                         /* <div title="itsmy">,<div title="myCat">, <div
                       title="its my cat">,<div title="...my...">*/
```

Рис. 6.10. **Фільтри за атрибутами**

Так само варто окремо відзначити наступний фільтр (рис. 6.11).

```
$("a[rel~='external']"); /* вибір всіх A з атрибутом rel, якій містить external в списку значень, розділених символом */

// У результаті його роботи будуть обрані наступні теги:
<a href="" rel="external">link</a> — так
<a href="" rel="nofollow external">link</a> — так
<a href="" rel="external nofollow">link</a> — так
<a href="" rel="friend external follow">link</a> — так
<a href="" rel="external-link">link</a> — ні
```

Рис. 6.11. **Фільтри за значенням атрибута**

Для роботи з формами та їх елементами є ряд селекторів, який дозволяє вибирати за типом елемента та фільтрів – enabled/disabled/ selected/checked (рис. 6.12).

```
$(":text");                          // вибір всіх input- елементів з типом =text
$(":radio");                         // вибір всіх input- елементів з типом =radio
                                     // і так далі
$("input:enabled");                  // вибір всіх включених елементів input
$("input:checked");                  // вибір всіх відзначених чекбоксів
$("form select[name=city] option:selected").val(); /* отримання обраного – их елементів в СЕЛЕКТ city */
$("form :radio[name=some]:checked").val();  /* отримання обраного значення радіобатона з ім'ям some */
$("form :checkbox:checked");         // вибір всіх обраних чекбоксів
```

Рис. 6.12. **Фільтри для роботи з формами та їх елементами**

Фільтри теж можна групувати:

```
$("div[name=city]:visible:has(p)"); /* вибір видимого div'а з ім'ям city, який містить тег p */
```

Варто звернути увагу на роботу з елементом <select>. Для цього існує ряд зручних прийомів, описаних в [15].

Для кращого завоєння матеріалу з бібліотеки jQuery слід розглянути декілька прикладів, які найбільш яскраво демонструють її можливості.

Висувна панель

За прикладом обрана слайд-панель, яка рухатиметься вгору/вниз кліком на посилання.

Після кліку на посилання буде перемикатися її клас (між "active" і "btn-slide"), а панель з id = "panel" висуватиметься/ховатися. Клас "active" змінює позицію фонового зображення (див. CSS-файл у додатку). Даний приклад поданий на рис. 6.13.

Магічні зникнення

Цей приклад покаже, як можна ефектно прибирати (розчиняти) елементи.

При натисненні на малюнок <img class = "delete"> буде знайдений батьківський елемент , і його прозорість буде повільно змінюватися від opacity= 1.0 до opacity=hide (см. рис. 6.14).

```
$(document).ready(function(){
        $(".btn-slide").click(function(){
              $("#panel").slideToggle("slow");
              $(this).toggleClass("active");
         });
});
```

Рис. 6.13. **Висувна панель**

```
$(document).ready(function(){
        $(".pane .delete").click(function(){
           $(this).parents(".pane").animate({ opacity: "hide" }, "slow");
        });
});
```

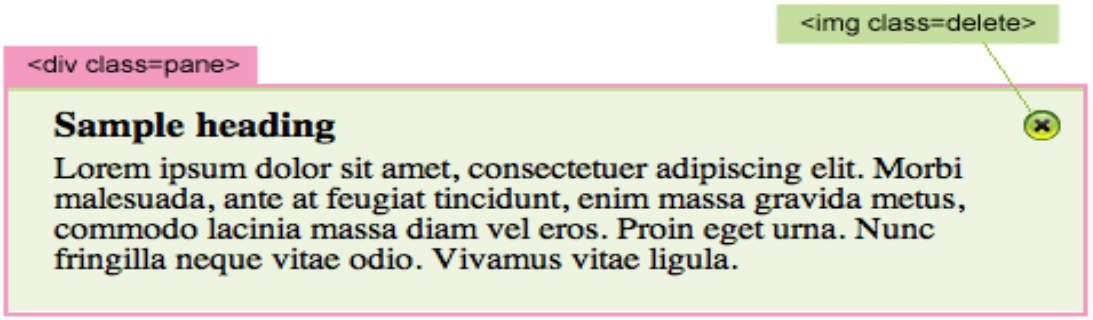


Рис. 6.14. **Магічні зникнення**

Пов'язана анімація

Більш складний приклад допоможе краще зрозуміти jQuery. Усього кілька рядків коду змусять квадрат рухатися, змінювати розмір і прозорість (рис. 6.15).

```
$(document).ready(function(){
        $(".run").click(function(){
         $("#box").animate({opacity: "0.1", left: "+=400"}, 1200)
         .animate({opacity: "0.4", top: "+=160", height: "20", width: "20"}, "slow")
         .animate({opacity: "1", left: "0", height: "100", width: "100"}, "slow")
         .animate({top: "0"}, "fast")
         .slideUp()
         .slideDown("slow")
         return false;
        });
});
```

Рис. 6.15. **Пов'язана анімація**

Коментар до коду з рис. 6.15:

завантажити сторінку (DOM готовий до маніпуляцій);

прив'язатись до події click для елемента <a class="run">;

маніпуляція елементом для зменшення його прозорості до 0.1, нарощування позиції left ще на 400 px зі швидкістю 1 200 (milliseconds);

повільно змінити наступні параметри: opacity = 0.4, top = 160 px, height = 20, width = 20; швидкість анімації вказується параметром "slow", "normal", "fast" або в мілісекундах;

далі opacity = 1, left = 0, height = 100, width = 100, швидкість – "slow";

далі top = 0, швидкість – "fast";

далі slideUp (з дефолтною швидкістю анімації – "normal");

далі slideDown, швидкість – "slow";

повернути false, щоб браузер не перейшов за посиланням.

Гармошка #1

Приклад реалізації "гармошки" наведено на рис. 6.16.

```
$(document).ready(function(){
        $(".accordion h3:first").addClass("active");
        $(".accordion p:not(:first)").hide();

        $(".accordion h3").click(function(){

          $(this).next("p").slideToggle("slow")
          .siblings("p:visible").slideUp("slow");
          $(this).toggleClass("active");
          $(this).siblings("h3").removeClass("active");
         });

});
```

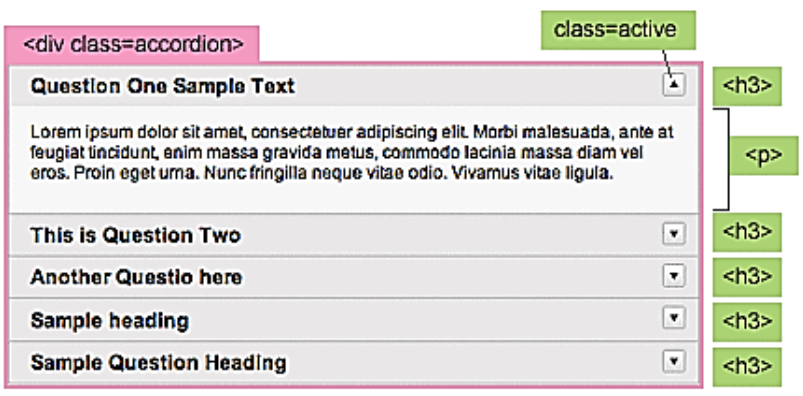


Рис. 6.16. **Гармошка #1**

Тепер час приступити до "розбору польотів".

Першим рядком додається клас "active" до першого елемента <h3> всередині (клас "active" відповідає за позиціонування фонового малюнка – іконки зі стрілочкою). На другій сходинці приховуються всі не перші <p> – елементи всередині .

Коли відбувається клік по заголовку <h3>, для наступного в ньому елемента <p> буде застосований ефект slideToggle, далі для всіх інших елементів <p> буде застосований ефект slideUp. Наступні дії змінюють клас заголовка на "active", далі відшукуються всі інші заголовки <h3> і прибирається у них клас "active".

Гармошка #2

Цей приклад схожий з попереднім, але відрізняється вказуванням відкритої за замовчуванням панельки (рис. 6.17).

```
$(document).ready(function(){

        $(".accordion2 h3").eq(2).addClass("active");
        $(".accordion2 p").eq(2).show();

        $(".accordion2 h3").click(function(){
            $(this).next("p").slideToggle("slow")
            .siblings("p:visible").slideUp("slow");
            $(this).toggleClass("active");
            $(this).siblings("h3").removeClass("active");
        });

});
```

Рис. 6.17. **Гармошка #2**

У CSS вказано для всіх елементів <p> display: none. Тепер необхідно відкрити третю панель. Для цього можна написати наступне $ (". Accordion2 p"), Eq (2), show (), де eq позначає рівність. Необхідно пам'ятати, що індексування починається з нуля.

Анімація для події hover #1

Даний приклад допоможе створити чудову анімацію для події hover (рис. 6.18).

Після наведення курсору миші на елемент меню (mouseover) відбувається пошук наступного елемента і анімується його прозорість і розташування.

```
$(document).ready(function(){
        $(".accordion h3:first").addClass("active");
        $(".accordion p:not(:first)").hide();

        $(document).ready(function(){
        $(".menu a").hover(function() {
            $(this).next("em").animate({
      opacity: "show", top: "-75"}, "slow");
        }, function() {
            $(this).next("em").animate({
                    opacity: "hide", top: "-85"}, "fast");
        });
});
```


```
<ul class="menu">
    <li>
        <a href="http://www.webdesignerwall.com">Web Designer Wall</a>
        <em>A wall of design ideas, web trends, and tutorials</em>
    </li>
    <li>
        <a href="http://bestwebgallery.com">Best Web Gallery</a>
        <em>Featuring the best CSS and Flash web sites</em>
    </li>
    <li>
```

Рис. 6.18. **Анімація для події hover #1**

6.4. Завдання до лабораторної роботи

1. Використовуючи бібліотеку jQuery, реалізувати завдання з лабораторної роботи № 5.

2. Порівняти рішення завдання на JavaScript з використанням бібліотеки jQuery та без неї і зробити висновки.

3. Проаналізувати можливості застосування бібліотеки jQuery та запропонувати (на вашу думку) мінімум три завдання, які найбільше підходять для ефективного їх вирішення за допомогою jQuery.

6.5. Контрольні запитання

1. Що таке JavaScript-биібліотека?
2. Назвіть відомі вам JavaScript-бібліотеки.
3. Опишіть загальну схему застосування jQuery.
4. Що спільного у CSS і jQuery?

Використана література

1. Будилов В. А. Основы программирования для Интернета / В. А. Будилов. – СПб. : БХВ-Петербург, 2003. – 736 с.
2. Глушков С. В. Программирование Web-страниц / С. В. Глушков, И. А. Жакин, Т. С. Хачиров. – Х. : Фолио, 2005. – 390 с.
3. Дарнелл Р. JavaScript: справочник / Р. Дарнелл. – СПб. : "Питер", 2001. – 192 с.
4. Калиновский А. И. Юзабилити: как сделать сайт удобным / А. И. Калиновский. – Мн. : Новое знание, 2005. – 220 с.
5. Лещев Д. Создание интерактивного web-сайта : учебный курс / Д. Лещев. – СПб. : Питер, 2003. – 544 с.
6. Ломов А. Ю. HTML, CSS, скрипты: практика создания сайтов / А. Ю. Ломов. – СПб. : БХВ-Петербург, 2006. – 416 с.
7. Мейер Эрик А. CSS-каскадные таблицы стилей: подробное руководство / Эрик А. Мейер. – М. : Символ, 2006. – 576 с.
8. Прохоренок Н. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. Прохоренок. – СПб. : БХВ-Петербург, 2008. – 640 с.
9. Создание Web-страниц и Web-сайтов. Самоучитель / под ред. В. Н. Печникова. – М. : Изд-во "Триумф", 2006. – 464 с.
10. Соколов С. А. HTML и CSS в примерах, типовых решениях и задачах. Профессиональная работа / С. А. Соколов. – М. : Вильямс, 2007. – 416 с.
11. Шмитт К. CSS. Рецепты программирования / К. Шмитт. – СПб. : "БХВ-Петербург", 2007. – 592 с.
12. Довідкове керівництво по HTML, CSS [Електронний ресурс]. – Режим доступу : http://htmlbook.ru.
13. Довідник по Web-мовам [Електронний ресурс]. – Режим доступу : http://www.spravkaweb.ru.
14. HTML довідник [Електронний ресурс]. – Режим доступу : http://html.manual.ru.
15. Internet інститут інформаційних технологій [Електронний ресурс]. – Режим доступу : http://www.intuit.ru.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Огурцов Віталій Вячеславович
Гриньов Денис Валерійович
Щербаков Олександр Всеволодович

ОСНОВИ ВЕБ ТА ВЕБ-ДИЗАЙН, ПРОГРАМУВАННЯ НА БОЦІ КЛІЄНТА

**Лабораторний практикум
з навчальної дисципліни
"Веб-технології та веб-дизайн"
для студентів напряму підготовки
6.050101 "Комп'ютерні науки"**

Відповідальний за випуск *Пономаренко В. С.*

Відповідальний редактор *Оленич М. М.*

Редактор *Ганцевич Н. І.*

Коректор *Маркова Т. А.*

План 2015 р. Поз. № 39-П.
Підп. до друку 05.11.2015 р. Формат 60×90 1/16. Папір офсетний. Друк цифровий.
Ум. друк. арк. 13,0. Обл.-вид. арк. 16,25. Тираж 400 пр. Зам. № 194.

Видавець і виготівник – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Леніна, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру
ДК № 4853 від 20.02.2015 р.