Тема: Інтерактивні WEB-документи Розробка динамічних елементів Web-сайтів з використанням JavaScript-сценаріїв.

Мета: Ознайомити з мовою JavaScript. Навчити правилам введення і виведення даних, використання типів даних, опису об’єктів і функцій JavaScript. Виховати навички користування JavaScript-словниками.

План

1. Загальний огляд технологій створення інтерактивних Web-документів
2. Основи мови програмування Javascript
3. Основні області застосування мови JavaScript
4. Автоматичне генерування тексту сторінок
5. Обробка подій
6. Зміна вмісту веб-сторінок
7. Відстеження координат миші
8. Об'єкт window і події, прив'язані до таймера

1. Загальний огляд технологій створення інтерактивних Web-документів

Стандартна мова розмітки HTML дозволяє легко і швидко створювати Web- документи, які передаються мережею. Проте сторінки, завантажені у вікно браузера, є статичними. Користувач не може змінити їх вміст, не може «взаємодіяти» з ними. Ідеологія Інтернету вимагала динамічного, інтерактивного подання інформації.

Для надання динамічності HTML-сторінкам було запропоновано і реалізовано ряд технологій, які дозволяють створювати сторінки, що «реагують» на дії користувача. Одна з перших технологій заснована на сценаріях CGL (Common Gateway Interface — загальний стандарт шлюзів прийому-передавання документів). Під ***сценарієм,*** або ***скриптом***, зазвичай розуміють «спрощену програму», засіб для розв’язання простих задач. CGI-сценарій — це програма, яка ініціалізується на сервері при передаванні на

нього інформації з полів форми HTML. Він опрацьовує передані йому дані, динамічно формує HTML-сторінку і відсилає результат в браузер користувача. Ці програми створюються на будь-якій із доступних на сьогодні мов — починаючи з мови Basic і закінчуючи bat- файлами DOS, чи командами Unix. Найчастіше CGI-сценарії пишуть мовою програмування Perl, яка спеціально призначена для роботи з текстами. Недоліком цієї технології є суттєвий вплив на завантаження мережі: будь-який запит і відповідь займають ресурси мережі. При цьому на сервер для виконання якої-небудь дії можуть бути передані неправильні дані. Користувач замість очікуваної (можливо досить довгий час) інформації отримає лише повідомлення про неправильно введені дані.

Для уникнення таких ситуацій фірмою Netscape було розроблено спеціальну мову сценаріїв **JavaScript.** Програми, написані цією мовою, вбудовуються безпосередньо в документ HTML та інтерпретуються браузером. Основна ідея JavaScript — можливість перевірки правильності введених у HTML-форму даних, зміни значень атрибутів HTML-дескрипторів і властивостей середовища (вікна браузера) у процесі перегляду HTML-сторінки користувачем. На практиці це виражається в тому, що можна, наприклад, змінити колір фону сторінки, або змінити інтегрований у документ малюнок, відкрити нове вікно або видати попередження. При цьому не відбувається перезавантаження сторінки. Подібна технологія знімає навантаження на мережу.

Створюючи JavaScript, розробники за основу взяли об’єктно-орієнтовану мову Java. Хоча за синтаксисом, набором команд, зарезервованими словами вони дуже подібні, але це зовсім різні мови і технології. Вони орієнтовані на виконання різних завдань у мережі, але доповнюють одна одну при створенні складних мережевих комплексів. Технологія Java дозволяє створювати програми, які можуть працювати як окремі закінчені проекти, або ж як додатки (аплети), підключені до HTML сторінки. Кожен аплет повинен зберігатися на сервері у своєму окремому

файлі. JavaScript реалізує свою функціональність лише у складі сторінки HTML і написана цією мовою програма починає працювати тільки тоді, коли цю сторінку завантажити в браузер

Щоб веб-сторінка була інтерактивною, тобто могла взаємодіяти з користувачем, і динамічною, необхідно використовувати скрипти, або сценарії. Сценарій (script, скрипт) — це програма, написана спеціальною мовою програмування і вбудована в HTML-документ. Сценарії описують усі можливі дії над елементами HTML-документа під час взаємодії з користувачем (наприклад, реакцію на натискання кнопки миші, зміну вмісту сторінки залежно від певних дій користувача тощо).

2. Основи мови програмування Javascript

Стандартною мовою для веб-скриптів є JavaScript — мова програмування, яка дає змогу вбудовувати виконувані модулі в документи, написані в кодах HTML. Програму, створену мовою JavaScript, інтерпретує браузер під час завантаження документа, в який вміщено її код. Проте різні браузери сприймають різні її варіанти. Версія мови JavaScript від корпорації Маикрософт, що має назву JScript, є найближчою до стандарту. Браузер Microsoft Internet Explorer підтримує не лише JScript, а й ще одну мову скриптів — Visual Basic Script (VBScript).

За допомогою мови JavaScript, можна, наприклад, зробити так, щоб після клацання зображення лівою кнопкою миші воно змінювало свій вигляд. її засобами можна реалізувати й складнішу поведінку елементів сторінки, скажімо, змусити їх пересуватися з необхідною швидкістю і за бажаною траєкторією.

За допомогою веб-сценаріїв можна створити принципово новий інтерфейс користувача для своєї сторінки. Всі події, генеровані браузером, такі як клацання кнопок, модифікація полів форм і переміщення між сторінками, можна перехопити й обробити засобами JavaScript. Ця мова придатна для розв'язування рутинних завдань, таких як перевірка

достовірності даних, опрацювання форм, виконання дій над текстовими і числовими значеннями, тобто тих завдань, які не можна розв'язати за допомогою стандартних засобів мови HTML.

Основні області застосування мови JavaScript:

- динамічне створення документа HTML за допомогою скриптів;

- перевірка достовірності полів форм HTML до передавання їх на сервер;

- локальне введення інформації для керування програмою;

- надання користувачу можливості вибору операцій, виконуваних браузером;

- виведення повідомлень для користувача у діалогових вікнах;

- локальне опрацювання форм, введення інформації користувачем.

Щоб використовувати мову скриптів ефективно, необхідно орієнтуватися в об'єктній моделі HTML-документа.

Програмний код JavaScript можна помістити в документ HTML у три способи:

- ✓ окремі скрипти розмістити в тілі документа, там, де в їхньому використанні є потреба;
- ✓ скрипти (функції, оголошення об'єктів) розмістити у заголовній частині документа між тегами <HEAD>...</HEAD>, а використовувати їх у тілі документа;
- ✓ зберегти скрипт у файлі (зазвичай із розширенням .js), а в документі дати посилання на нього

Javascript - мова програмування, яка застосовується в основному для розробки клієнтської частини сайтів (**front-end**).

Javascript є інтерпретатором і виконується усередині браузера (програми, яка візуалізує веб-сторінку). Браузер спочатку аналізує текст програми, інтерпретує його, і тільки потім виконує потрібні дії. Це повільно. Тому програми на Javascript працюють повільно й не завжди однаково для різних браузерів.

Чому тоді інтерпретатор, а не компілятор? Мова Javascript був спеціально розроблений як інтерпретатор, щоб була можливість виключити з нього всі операції роботи з файлами (читання, запис і т.п.). Це необхідно для того, щоб веб-сторінка, завантажена з мережі Інтернет і утримуюча програму Javascript, не могла зашкодити комп'ютеру користувача (наприклад, заразити його вірусом). Якщо зараження вірусом і відбувається, то через якісь інші причин, але не з вини Javascript.

Отже, писати й читати файли програма на Javascript не може. А що ж вона може? Для чого в основному використовується Javascript? Javascript призначений для:

1) Автоматизації процесу набору тексту сторінок;

2) Обробки подій, що відбуваються на сторінці (додання сторінці інтерактивності);

3) Зміни вмісту веб-сторінок під час їх перегляду, аналіз даних веб-сторінок;

4) Формування запитів до сервера й одержання відповідей без перевантаження веб-сторінки;

5) Робота з cookies (дані веб-сторінки, що зберігаються в браузері на комп'ютері користувача).

Нічого із цього не можна добитися за допомогою HTML і CSS, тому Javascript доповнює ці інструменти веб-програмування, а не заміняє або дублює їх.

Перед тем, як приступитися до опису Javascript, відзначимо, що тут будуть розглянуті лише необхідні основи, достатні для того, щоб почати програмувати на Javascript. Усе інше ви зможете вивчити самостійно в міру практичної потреби. Ми тут не будемо зупинятися на перерахуванні конструкцій і типів мови програмування Javascript, а також докладному описі його синтаксису. Будемо вважати, що ви вже знаєте який-небудь із мов програмування, наприклад, C/C++. У цьому випадку ви легко зрозумієте синтаксис Javascript, а основне знання, яке вам буде потрібно придбати, це як за допомогою Javascript вирішувати його основні завдання, перераховані вище. Цьому й конкретним прикладам ми й присвятимо основний обсяг даної лекції.

3. Автоматичне генерування тексту сторінок

Щоб почати програмувати на Javascript не потрібно нічого, крім одного (кожного) із браузерів Інтернету, установленого на нашім комп'ютері. Текст програми на Javascript вставляється прямо в Html-Сторінку усередині тегу <**script**> .. </**script**>. Таких тегів може бути трохи й браузер виконує поміщені усередині них шматки Javascript-програм відразу ж, як тільки доходить до цього місця в процесі завантаження веб-сторінки.

Розглянемо текст простої веб-сторінки, який показано на рисунку 1 (файл ex3_01.html).

```
<html>
<body>
Перша програма на Javascript.<BR>
<script type="text/javascript">
document.writeln('Hello, digital world!');
</script>
</body>
```

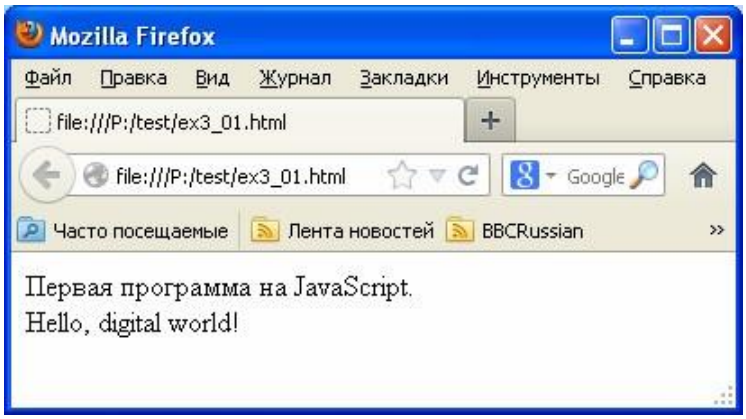

Рисунок 1 - Приклад простої сторінки з Javascript

Атрибут **type** тегу <**script**> у цьому випадку повідомляє браузер, що він має справу з Javascript (теоретично тут може бути скрипт на іншій мові програмування). Однак для всіх браузерів мовою програмування за замовчуванням є саме Javascript. Тому замість рядка <script type="text/javascript"> можна було б написати от так: <script>

Функція document.writeln('Hello, digital world') виводить напис «Hello, digital world» на **document** (цим ключовим словом на Javascript позначається вся веб-сторінка, у яку вставлений скрипт).

Тут **document** - це об'єкт Javascript, а **writeln** через крапку від **document** - функція. Крапка між ними вказує, що функція належить цьому об'єкту й, у даному конкретному випадку виводить текст на цей об'єкт.

Напис у даному прикладі виводиться в тому місці сторінки, де був вставлений скрипт.

Вищенаведений приклад просто ілюструє спосіб вставки програми на Javascript на веб-сторінку. Взагалі ж вставити цей напис на веб-сторінку можна було б і засобами HTML. Розглянемо більш складний приклад, у якім при спробі обійтися засобами HTML нас би чекав серйозний обсяг роботи, у той час як програма на Javascript робить усі автоматично.

Нехай потрібно вивести на веб-сторінку 200 чисел, відзначаючи прості числа (які ні на що не діляться) червоним кольором. Програма на Javascript

буде містити функцію, що повертає 1, якщо задане число - простої, а також умовні оператори й оператори циклу (подібні відповідних конструкцій інших мов програмування).

На рисунку 2 наведена така програма (файл ex3_02.html).

```
<html>
<body>
Прості числа виділені червоним кольором<BR>
<style>
.myred { color:red; }
</style>
<script> function prost(n)
{ for (i=2;i<n;i++) if ((n % i)==0) return 0; return 1;
} for (papa=1;papa<=200;papa++) if (prost(papa)==0) document.writeln(papa+' '); else document.writeln('<span class="myred"> '+papa+' </span>');
</script>
</body>
```

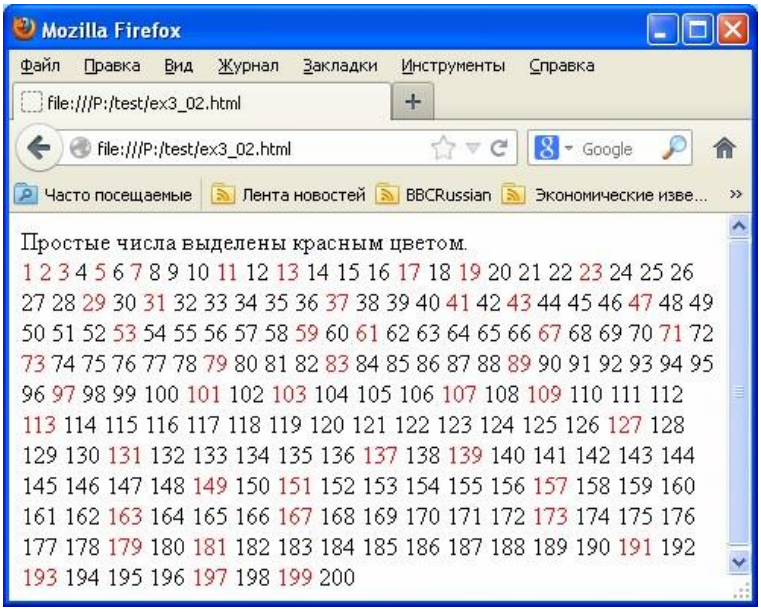

Рисунок 2 - Програма на Javascript, яка відзначає прості числа червоним кольором

Функції на Javascript схожі на функції C/C++ за тим виключенням, що не потрібно повідомляти тип вхідних змінних і тип результату функції. Браузер, який виконує скрипт, за значенням, що присвоюється, сам зрозуміє, що мА.ться на увазі.

Дана функція перевіряє число n на простоту. Для цього в циклі **for** вона знаходить залишки від розподілу n на всі числа від 2 до n-1 (оператор **%**). Якщо хоч один їхніх цих залишків дорівнює нулю, то функція повертає (оператор **return**) 1. Інакше (після виходу із циклу) функція повертає 0 (ознака того, що число - простої).

Основна програма за допомогою циклу **for** (змінна “papa” названа так, щоб показати, що імена змінних ми можемо вибирати такі, які нам подобаються) і функції prost(papa).

Не прості числа виводяться на веб-сторінку без усякого оформлення (оператор **document.writeln**).

Прості числа виводяться на веб-сторінку, обрамлені тегом <**span**> .. </**span**>, у якого виставлений атрибут class=“myred”. Стиль “.myred” у свою чергу був описаний вище в тегу <**style**> .. </**style**>. У нього прописана єдина властивість - “color:red;” (червоний колір). Тому всі прості числа виводяться на веб-сторінку червоним кольором.

Оборотний увага також на оператор “+”, який використовується в програмі для об'єднання строкових змінних. Якщо цим оператором складаються два числа, то результатом буде їхня сума. Якщо ж складають два рядки (або число й рядок), то результатом буде рядок. Наприклад, у результаті виконання коду x=57; s=‘Nikolay’+x; у змінній s буде втримуватися рядок ‘Nikolay57’.

Наведений на рисунку 5.2 приклад ілюструє ситуацію, коли вручну набирати текст сторінки набагато довше, чим написати програму. Частіше ж буває, що сторінка містить якісь змінні дані й повинна щораз виглядати по іншім. Javascript у даній ситуації незамінний.

4. Обробка подій

Те, заради чого Javascript і був створений - це обробка подій на веб-сторінках, наприклад, натискань мишкою на якісь елементи сторінки. На рисунку 3 наведений приклад простої сторінки із кнопкою (файл ex3_05.htm).

```
<html>
<body>
Це приклад кнопки: <br><br>
<button>Кнопка</button>
</body>
```

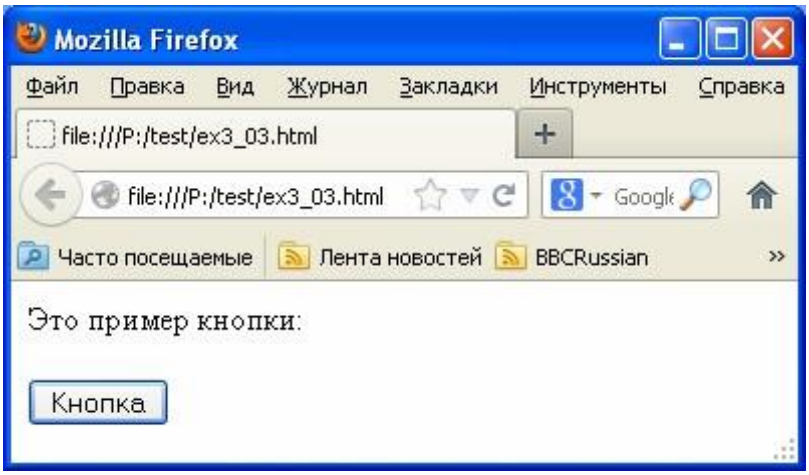

Рисунок 3 - Веб-сторінка із кнопкою

Тут ми бачимо новий для нас тег <**button**> .. </**button**>, який додає на веб-сторінку кнопку. Усередині тегу може бути вставлений якийсь напис, а може, наприклад, бути вставлене й зображення (тег <**img**>).

Ця кнопка нічого не робить. Можна на неї натиснути (вона буде натиснута), але при цьому нічого не відбудеться. Щоб щось відбулося, потрібно використовувати Javascript.

У тегу <**button**>, як і в більшості інших тегів, є спеціальні атрибути, призначення яких - обробка подій. За натискання мишкою відповідає атрибут

onclick. У якості значення цього атрибута повинен бути вставлений якийсь текст на Javascript приблизно от так:

<button onclick=”Текст Javascript”>

У цьому випадку при натисканні на дану кнопку буде виконаний вставлений текст програми.

На рисунку 4 наведений текст цієї ж веб -сторінки з обробкою натискань на кнопку (файл ex3_04.html).

```
<html>
<body>
Це приклад кнопки: <br><br>
<button onclick="alert('Кнопка натиснута')">Кнопка</button>
</body>
```

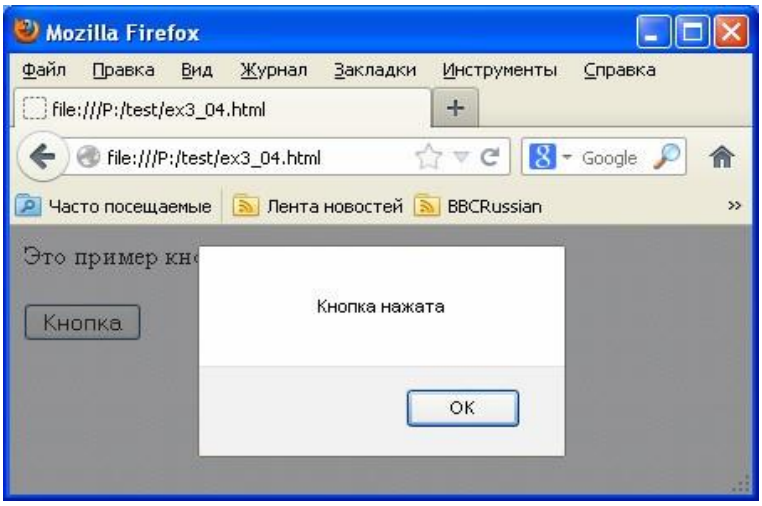

Рисунок 4 - Веб-сторінка із кнопкою й обробкою події onclick

Тут у якості тексту програми зазначене: alert('Кнопка натиснута')

Функція **alert** є стандартною функцією Javascript і видає на екран вікно з написом (див. Рисунок 4).

Давайте розглянемо ледве більш складний приклад, щоб показати, що програма, що реагує на подію, може складатися з багатьох команд і взаємодіяти з іншими частинами Javascript на цій веб-сторінці.

Веб-сторінка на рисунку 5 підраховує, скільки раз ми нажали на цю кнопку (файл ex3_05.html) і видає результат підрахунків усередині спливаючого вікна за допомогою функції **alert**.

```
<html>
<body>
<script> var n=0;
</script>
Це приклад кнопки: <br><br>
<button onclick="n=n+1;alert('Число натискань кнопки:
'+n)">Кнопка</button>
</body>
```

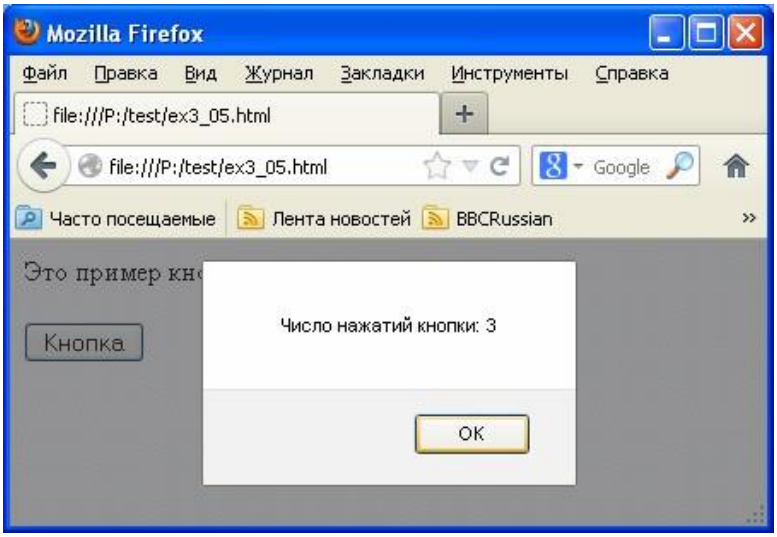

Рисунок 5 - ще одна веб-сторінка із кнопкою й обробкою події onclick

Тут по натисканню кнопки виконується наступний текст програми:

n=n+1;

alert('Число натискань кнопки: '+n)

У змінній n ми зберігаємо число натискань кнопки. При завантаженні сторінки в тегу <**script**> .. </**script**> ми оголосили цю змінну (ключове слово **var**) і привласнили їй нуль. Далі при кожнім натисканні її значення збільшується на одиницю.

Оборотний увага, що писати при об’яві змінних ключове слово **var** рекомендується, але не обов'язково (браузери розуміють і так). Зокрема, замість тексту

var n=0;

можна було написати просто

n=0;

До речі, ставити крапку з коми після кожної команди на Javascript теж не обов'язково. Але краще це робити. Якщо текст програми обробки події натискання на кнопку буде довгим, то він порядком утруднить розуміння тексту веб-сторінки, якщо поміщати його усередину атрибута **onclick**. Краще виносити його в окрему функцію. Текст веб-сторінки буде ледве довше, але при цьому буде виглядати акуратніше й зрозуміліше (див. Рисунок 6, файл ex3_06.html).

```
<html>
<body>
<script>
var n=0;
function misha()
{
n=n+1;
alert('Число натискань кнопки: '+n);
}
</script>
Це приклад кнопки: <br><br>
<button onclick="misha()">Кнопка
</button>
</body>
```

Рисунок 6 - Обробка натискання на кнопку винесена в окрему функцію misha

Як бачимо, усередині атрибута **onclick** залишився тільки виклик функції “misha”, а текст самої функції перебуває в іншім місці веб-сторінки усередині тегу <**script**> .. <**script**>.

Розглянемо тепер приклад обробки подій не для кнопки, а якого-небудь іншого тегу HTML, наприклад, <**div**> (див. Рисунок 7, файл ex3_07.html).

```
<html>
<body>
<div style="width:200px; height:100px; background-color:lightgreen;"
onclick="alert('Натиснуте')">Текст</div>
</body>
```

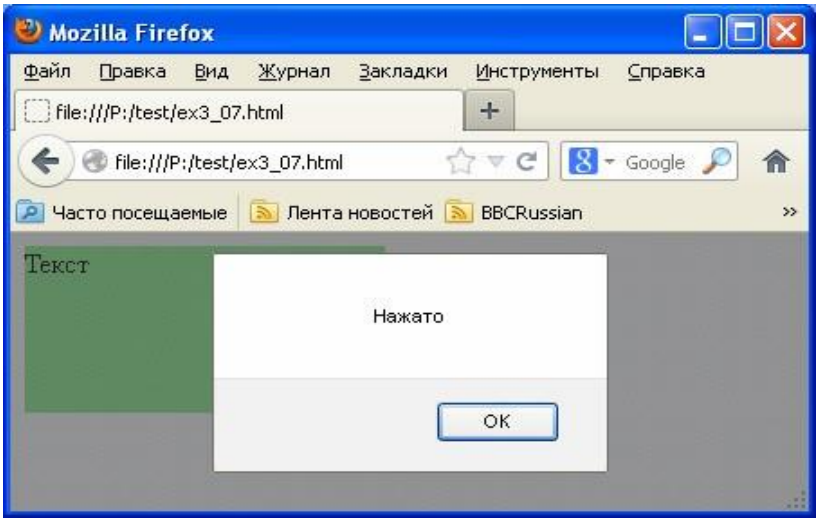

Рисунок 7 - Обробка натискання на прямокутник, заданий тегом <**div**> .. </**div**>

Як бачимо, зроблене всі точно так само, як і для тегу <**button**>. При натисканні на елемент (зелений прямокутник) вискакує віконце з написом. Точно так само атрибут **onclick** можна встановлювати для більшості видимих тегів HTML.

А які ще є атрибути тегів, відповідальні за обробку подій? Перелічимо найбільше часто використовувані з них:

onblur:	втрата фокуса;
onchange:	зміна значення елемента форми;
onclick:	клацання лівою кнопкою миші на елементі;
ondblclick:	подвійне клацання лівою кнопкою миші на елементі;
onfocus:	одержання фокуса;
onkeydown:	клавіша натиснута, але не відпущена;
onkeypress:	клавіша натиснута й відпущена;
onkeyup:	клавіша відпущена;
onload:	документ завантажений;
onmousedown:	натиснута ліва кнопка миші;
onmousemove:	переміщення курсору миші;
onmouseout:	курсор залишає елемент;
onmouseover:	курсор наводиться на елемент;
onmouseup:	ліва кнопка миші відпущена;
onreset:	форма очищена;
onselect:	виділений текст у поле форми;
onsubmit:	форма відправлена;
onunload:	закриття вікна.

Розберіться із цими атрибутами самостійно (практичним шляхом і за допомогою довідників), а тут ми розглянемо ще один приклад, який дозволить нам краще розуміти поведінка програми на Javascript (Рисунок 8, файл ex3_08.html).

```
<html>
<body> <button onclick="more()">Кнопка</button>
<script> more(); </script>
<br>Тут якийсь текст<br>
<script> function more()
{
alert('Мope!');
}
</script>
</body>
```

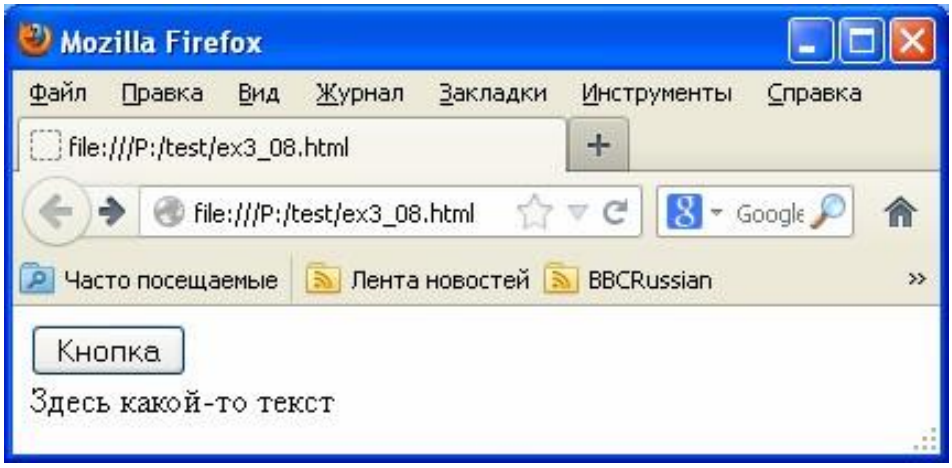

Рисунок 8 - Ілюстрація до особливостей розміщення тексту Javascript на веб-сторінці

Спробуйте відкрити в браузері цю сторінку. Вона намалює кнопку, текст, але нічого не виведе на екран. Але ж у тексті цієї веб-сторінки викликається функція more():

<script> more(); </script>

При виклику цієї функції по ідеї повинне спливти віконце з написом “Море”. Але не спливе.

А ще на сторінці є кнопка, при натисканні на яку теж викликається функція more(). Спробуйте нажати на неї. А зараз вікно з написом море з'явилося! Чому? Чому в одному випадку функція не спрацювала, а в іншому - спрацювала?

Тому що браузер намагається виконати Javascript відразу ж, як тільки його зустрічає (прямо в процесі завантаження сторінки). І в той момент, коли браузер частково завантажить текст сторінки й побачить команду:

<script> more(); </script>

інша частина сторінки ще не буде завантажена. І в тому числі ще не буде завантажена та частина сторінки, де оголошена функція more(). Тому браузер проігнорує виклик цієї функції (він такої функції поки не знає).

У той же момент, коли ми натискаємо на кнопку, сторінка вже повністю завантажена, браузер уже завантажив те місце веб-сторінки, де оголошена more() і тому може викликати цю функцію.

Ураховуйте цей нюанс при розробці своїх сторінок. Поки веб-сторінка не завантажилася повністю, не можна з

Javascript звертатися до тем шматкам програми, які вставлені в сторінку далі по тексту. Іноді для перевірки завантаженості сторінки буває зручно використовувати подію **onload** (документ завантажений). Його можна прив'язати до тегу <**body**> .. </**body**> (уся веб-сторінка), наприклад, як це показано на рисунку 9 (файл ex3_09.html).

```
<html>
<body onload="more()">
Тут якийсь текст
<script> function more() { alert('Mope!');}
</script>
</body>
```

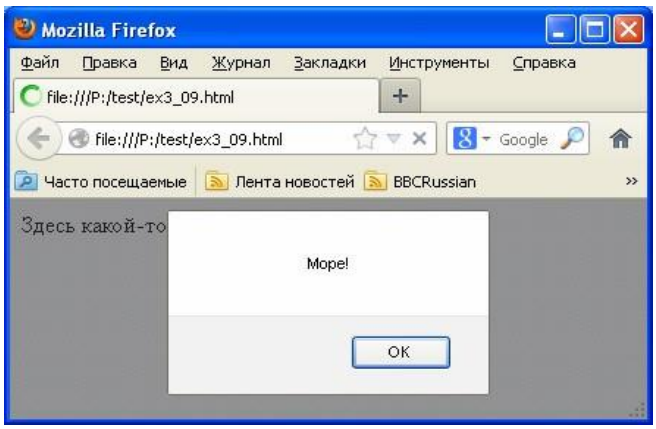

Рисунок 9 - Ілюстрація використання події **onload**

5. Зміна вмісту веб -сторінок

Об'єкт document має ряд корисних функцій, наприклад, **getelementbyid**(id), які дозволяють одержати доступ до елементів веб-сторінки й міняти їхній зміст. На рисунку 10 наведений приклад використання цієї функції (файл ex3_10.html).

```
<html>
<body>
<style> div { background-color:yellow; padding:15px;
}
</style>
<script> function primer()
{ s=document.getelementbyid('olga');
s.innerhtml='Мене переїхали!';
}
</script>
Приклад зміни тексту
<div id="olga" onmouseover="primer()">
Жовтий прямокутник </div>
</body>
```

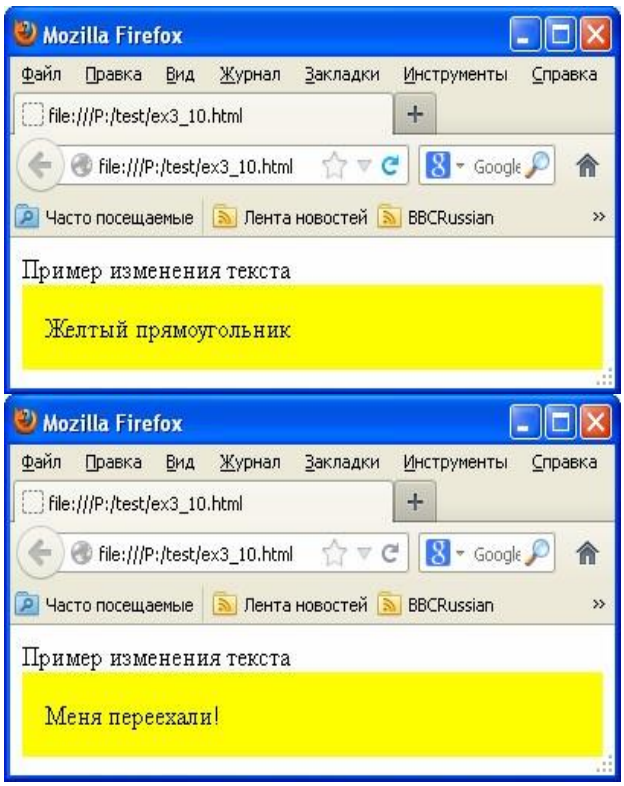

Рисунок 10 - Приклад використання події **onload**

Якщо навести мишку на жовтий прямокутник (подія **mouseover**), то викликається функція primer(). Ця функція знаходить елемент веб-сторінки, у якого id=”olga” (за допомогою функції **getelementbyid**(‘olga’)) і записує покажчик на цей елемент у змінну **s** (назва з однієї букви ми дали цієї змінної, щоб текст програми був більш коротким).

Далі ми заміняємо значення поля **innerhtml** (**innerhtml** - це весь текст веб-сторінки, який перебуває усередині даного тегу <**div**> .. </**div**>) на текст “Мене переїхали!”. У результаті замість напису “Жовтий прямокутник” з'явиться напис “Мене переїхали!”. За аналогією можна міняти текст сторінки, значення елементів сторінки, значення властивостей стилів. Алгоритм полягає всього із двох кроків: 1) Знайти елемент по його імені або id; 2) Указати, яке поле цього елемента потрібно поміняти й указати нове значення для нього.

<html>

<body>

<style> div { background-color:yellow; padding:15px;

}

</style>

<script> function primer()

{ s=document.getelementbyid('olga'); s.style.backgroundcolor='#ff8888'; if (s.innerhtml.length<25)

s.innerhtml=s.innerhtml+'
Мене переїхали!'; else

s.innerhtml=s.innerhtml+'
Мене знову переїхали!';

}

</script>

Приклад зміни тексту

Жовтий прямокутник

</body>

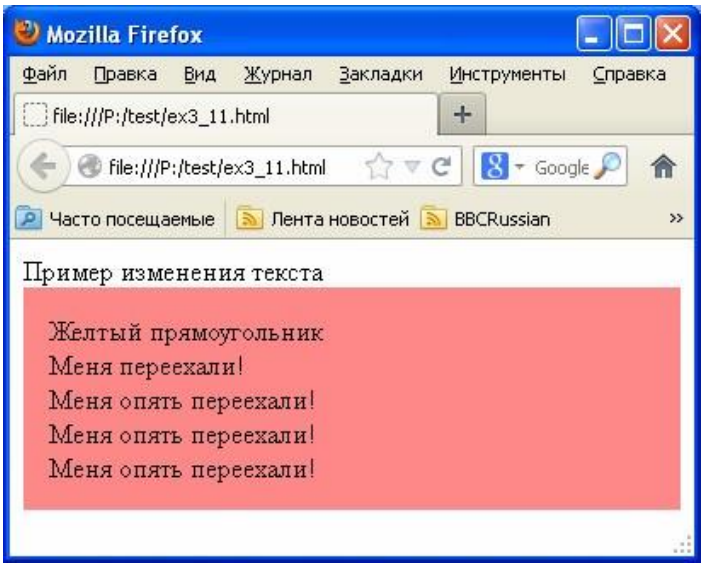

Рисунок 11 - Вікно браузера після декількох наведень курсору мишки на кольоровий прямокутник з текстом

На рисунку 11 наведений попередній приклад, але небагато доповнений: при наведенні мишки міняється колір прямокутника, а напис не заміщає старий текст, а додається до нього (файл ex3_11.html).

Тут по події ми міняємо колір тла прямокутника з жовтого на ясно-червоний (#ff8888). Для цього змінюється поле **backgroundcolor** поля **style** знайденого по id елемента **s** веб-сторінки. Будь-яка інша властивість стилю можна міняти по цій же схемі:

змінна.style.властивість = значення;

Наприклад, можна було б написати так:

s.style.padding=’25px’;

або

s.style.fontfamily=’Verdana’;

Зверніть увагу, що властивості стилів в CSS називаються “**background-color**” і “**font-family**”, а в Javascript вони називаються “**backgroundcolor**” і

“**fontfamily**”. Це через те, що в назвах змінних і полів в Javascript не можна використовувати знак “-“ (якщо написати font-family, те браузер подумає, що ми прагнемо зі значення змінної font відняти значення змінної family). Загальне правило перекладу назв властивостей з CSS в Javascript таке: викидаємо знак “-“, а наступну після нього букву робимо великою. Наприклад, якщо нам буде потрібно поміняти значення властивості **backgroundposition**, те в Javascript потрібно писати **backgroundposition**. Дописати текст замість заміщення просто.

Замість s.innerhtml=’новий текст’;

ми пишемо s.innerhtml= s.innerhtml+’новий текст’;

І останнє пояснення до цього прикладу. Поле length будь-якого рядка містить її довжину. Наприклад, якщо ми напишемо так:

s='Nikolay';

u=s.length;

те в змінній u буде перебувати число 7. У нашім наступному прикладі ми розглянемо подію **onchange** і спробуємо змінити атрибут **value** тегу <**input**> HTML.

Тег <**input**> ставиться до елементів форм HTML, які докладніше ми розглянемо на наступній лекції разом з основами PHP. Тут же нам буде потрібно два рядки введення, які можна організувати, якщо тегу <**input**> задати значення атрибута type=”**text**”. Значення атрибута **value** тегу <**input**> буде при цьому містити поточне значення введеного рядка.

Наша веб-сторінка буде обчислювати факторіал заданого числа. В одну із двох рядків будемо вводити число, а в другий виводити факторіал цього числа.

Щоб довідатися, що користувач увів якесь число, будемо відслідковувати подія **onchange** (дані змінилися) для рядка, у який користувач уводить число. На рисунку 12 наведений текст цієї веб -сторінки (файл ex3_12.html) і результат уведення користувачем числа 7.

```
<html>
<script> function faktorial(n)
{ var k=1; for (var i=1;i<=n;i++) k=k*i; return k;
} function obrabotka()
{ n=document.getelementbyid("mama").value;
document.getelementbyid("otvet").value = faktorial(n);
}
</script>
<body>
Число: <input id="mama" onchange="obrabotka()"
type="text" value="1"></input> <br>
Факторіал: <input id="otvet" type="text" value="1"></input>
</body>
```

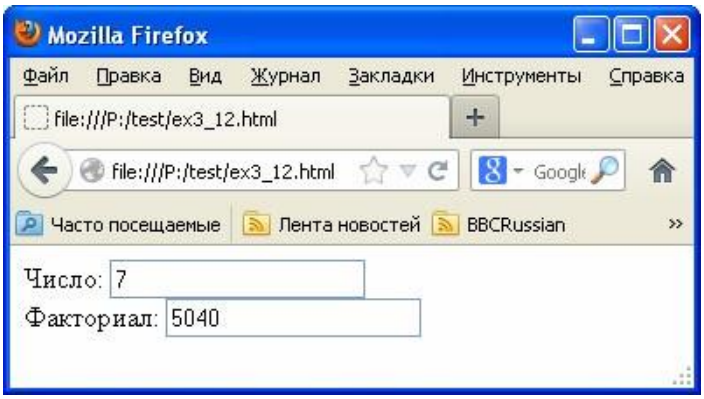

Рисунок 12 - Відстеження події **onchange** і зміна значень поля **value** елементів форми

Спробуйте запустити цю сторінку. Зверніть увагу, що подія виникає не відразу ж, після того, як ми кликнули на рядок уведення мишкою й нажали на клавіатурі цифру 7, а тільки після того, як ми пішли зі зміненого рядка введення (кликнули мишкою десь в іншім місці веб-сторінки або нажали клавішу “Enter”). Потрібно враховувати це при використанні події **onchange**.

6. Відстеження координат миші

Іноді при обробці події потрібно знати, де в цей момент перебував курсор мишки. Для цього потрібно не тільки знати про подію (русі мишки), але й знати його параметри (координати курсору мишки). Для цього потрібно перехопити подію руху мишки й призначити функцію, яка буде одержувати параметри події. У загальному виді ця процедура виглядає так:

document.captureevents(Event.Подія); document.onmousemove=функція;

Тут “подія” - назва тієї події, яка нам потрібно перехопити, а “функція” - назва нашої функції, яка буде обробляти подія.

На рисунку 13 наведена коротка веб -сторінка, що показує, як це зробити (файл ex3_15.html).

```
<html>
```

```
<body>
<script> function khai(eve)
{ s=document.getelementbyid('taras');
s.innerhtml='X='+eve.pagex+'
Y='+eve.pagey;
}
document.captureevents(Event.MOUSEMOVE);
document.onmousemove=khai;
</script>
<div id="taras"></div>
</body>
```

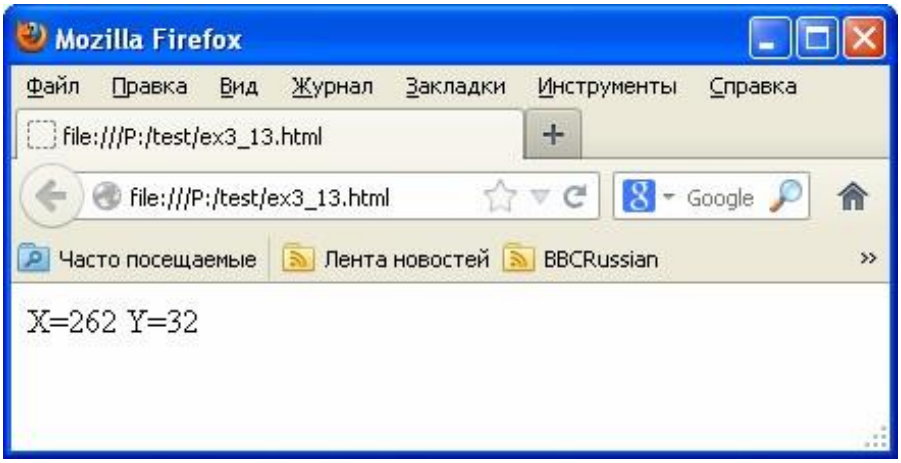

Рисунок 13 - Проста веб-сторінка з відстеженням координат курсору мишки

У даному прикладі ми перехопили подію **MOUSEMOVE**: document.captureevents(Event.MOUSEMOVE); і призначили функцію khai для його обробки: document.onmousemove=khai;

У функцію khai передається параметр **eve** (усі дані про подію), у якого є поля, що **цікавлять нас, eve.pagex** (координата X курсору мишки) і **eve.pagey** (координата Y курсору мишки).

Функція khai одержує доступ до вмісту прямокутника <**div**> з іменем “taras” s=document.getelementbyid('taras'); і виводить на нього отримані в eve координати курсору мишки: s.innerhtml='X='+eve.pagex+' Y='+eve.pagey;

Поберіть собі за основу цей приклад. А ми розглянемо боле складну веб-сторінку, де нам знадобляться всі наші знання (Рисунок 14, файл ex3_14.html).

```
<html>
<style>
.qq, .ima, .big { position: absolute; }
.qq, .ima { background:url('griv.png'); }
.qq, .big { border: 1px solid black; border-radius:15px; visibility:hidden; } .qq { top:9px;left:9px; cursor:crosshair; width:80;height:80; }
.ima { width:512;height:377; top:10px;left:10px }
.big { background:url('grivb.jpg'); top:10px;left:550px; width:320 ; height:320; }
</style>
<script> var posx=10; posy=10; Loy=377; Lox=512; function khai(eve)
{ mx=eve.pagex;my=eve.pagey; if ((mx>posx+40) & (mx<posx+ Lox40)&(my>posy+40)&(my<posy+Loy-40))
{ lup.top=my-41;lup.left=mx-41; lup.backgroundposition="-"+(mx-50)+"px -"+(my-50)+"px"; bol.backgroundposition="-"+((mx10)*4-160)+"px -"+((my-10)*4-160)+"px"; lup.visibility="visible"; bol.visibility="visible"; } else {lup.visibility="hidden"; bol.visibility="hidden"; } }
function myst()
```

```
{lup=document.getelementbyid('lupa').style;
bol=document.getelementbyid('uvel').style;
document.captureevents(Event.MOUSEMOVE);
document.onmousemove=khai; }
</script>
<body>
<div class=ima id="small"></div>
<div class=qq id="lupa"></div>
<div class=big id="uvel"></div>
<script> myst() </script>
</body>
```

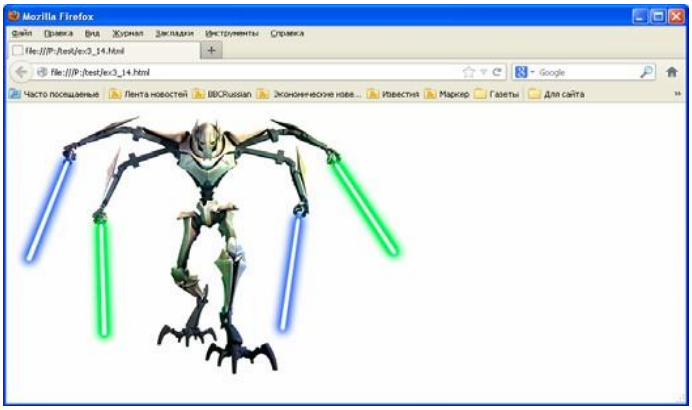

Рисунок 14 - Сторінка з масштабуванням зображення

Спробуйте самостійно розібратися в цьому тексті веб-сторінки. Усе, що тут використовується, ми вже знаємо. Анімація в цій веб-сторінці (при наведенні курсору на зображення праворуч з'являється його збільшений фрагмент, див. Рисунок 15) здійснюється за д опомогою зміни властивостей стилів з Javascript.

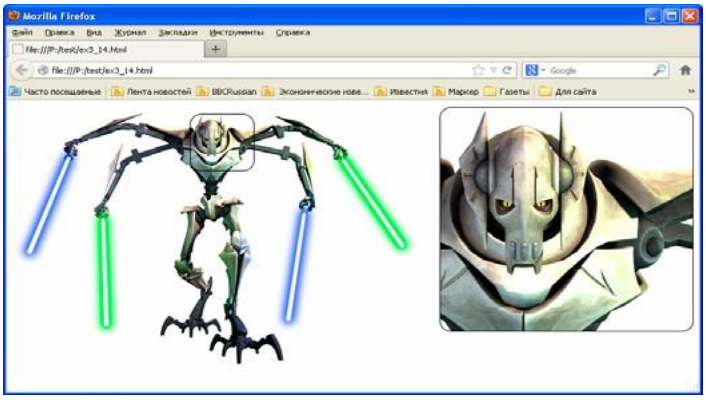

Рисунок 15 - При наведенні курсору мишки на зображення праворуч з'являється його збільшений фрагмент

Стиль “.qq” відповідає за лупу на зменшенім зображенні, стиль “.ima” - за зменшене зображення, а стиль “.big” - за збільшений фрагмент зображення, на який наведена лупа.

7 Об'єкт window і події, прив'язані до таймера

Ми вже не раз використовували стандартний для Javascript об'єкт **document**. Розглянемо ще один стандартний об'єкт - **window**.

Об'єкт **window** дає доступ до функцій браузера. Наприклад, можна відкрити нове вікно браузера (викликавши функцію **window.open**()) або закрити існуюче (викликавши функцію **window.close**()). Можна також міняти якісь властивості вікна браузера, наприклад, задати текст, який у цей момент виводиться в рядку статусу (**window.status**=’Текст, який потрібно вивести’). Розглянете методи й властивості об'єкта window самостійно в міру практичної необхідності. Можливо, на практиці Вам буде потрібно створювати свої власні об'єкти. За допомогою будь-якого довідника подивитеся, як це робити.

Останній тип подій, який ми тут розглянемо - це події часу (таймера). Щоб прив'язати функцію до таймера, потрібно в програмі написати:

Тут myfun - ім'я функції, яка буде викликатися один раз в 3000 мілісекунд (раз в 3 секунди), а m - змінна, яка буде потрібна, якщо ми захочемо виключити таймер. Таймер у цьому випадку вимикається командою:

window.clearinterval(m);

На рисунку 16 наведений п риклад веб-сторінки з обробкою подій часу файл ex3_15.html).

```
<html>
<body> <script> n=0; function ok()
{ n=n+1; s=document.getelementbyid('ua');
s.innerhtml=n+' sec'; } m=window.setinterval(ok,1000);
</script>
<div id="ua"></div>
</body>
```

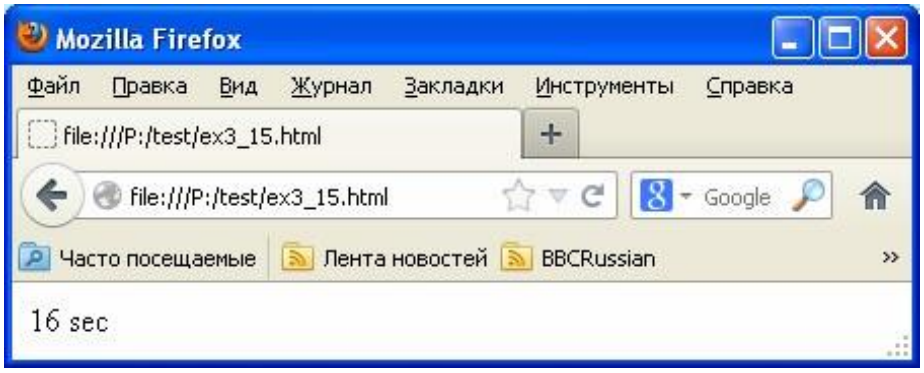

Рисунок 16 - Обробка подій таймера

На цьому дана лекція закінчується. Далі ви будете вивчати Javascript самостійно, засвоюючи новий матеріал у міру того, як це буде потрібно при розв'язку тієї або іншого практичного завдання.

На жаль, таких гарних довідників, як для HTML і CSS, для Javascript немає. Можна користуватися, наприклад, такими довідниками, як http://javascript.narod.ru або http://javascript.ru, але вони не дуже зручні, зрозумілі й самодостатні. Прийде часто використовувати пошук в Google,

наприклад от по таких запитах: “ як довідатися час із Javascript”, “ як згенерувати випадкове число на Javascript” і т.д. Так буде простіше знайти потрібну інформацію, чому в довіднику.

Пізніше (на лекції №6) ми розглянемо бібліотеку Jquery - набір функцій, написаних на Javascript і полегшуючих роботу з Javascript (обробку подій, анімацію, відправлення й одержання запитів до сервера, роботу з **cookies**).

Контрольні запитання

1. З якою метою розроблена мова JavaScript?
2. Порівняйте JavaScript та Java.
3. Які особливості роботи скриптових програм?
4. Як підключати сценарії до web-документа?
5. Які об’єкти входять до об’єктної моделі мови JavaScript?
6. Які типи даних характерні для JavaScript?
7. Як оголошуються змінні в JavaScript?
8. Що таке глобальна та локальна область видимості змінних?
9. Як задавати дату в JavaScript?
10. Які існують способи завдання функцій?
11. Що таке динамічна функція?
12. Що таке вбудовані функцій JavaScript?
13. Які базові події підтримуються в JavaScript?
14. Які способи створення оброблювачів подій?
15. Як підключити бібліотеку скриптів?
16. Що таке банер і як його створювати?
17. Як забезпечується сумісність з різними типами браузера?
18. Які базові події підтримуються в JavaScript та VBScript?