

Лабораторна робота № 1

Тема: **Розробка сценаріїв для веб-сторінок за допомогою скриптової мови JavaScript.**

Мета роботи: Метою роботи є практичне освоєння програмних засобів розробки сценаріїв для веб-сторінок за допомогою скриптової мови JavaScript.

Теоретичні відомості

JavaScript - це мова програмування, що використовується в складі HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачем. JavaScript є однією із складових динамічного HTML. Ця мова програмування була створена фірмами Netscape та Sun Microsystems на базі мови програмування Sun's Java. На сьогодні є декілька версій JavaScript. Однією із найбільш поширених є версія JavaScript 1.3.

За допомогою JavaScript на HTML-сторінці можливо зробити те, що не можливо зробити за допомогою стандартних тегів HTML.

Код програми JavaScript розміщується або в середині HTML-сторінки, або в текстовому файлі, що пов'язаний за допомогою спеціальних команд з HTML-сторінкою. Цей код, як правило, розміщується в середині тегу HTML та завантажується в браузер разом з кодом HTML-сторінки.

Програма JavaScript не може існувати самостійно, тобто без HTML-сторінки. Виконання програми JavaScript відбувається при перегляді HTML-сторінки в браузері, звичайно, тільки в тому випадку, коли браузер містить інтерпретатор JavaScript. Практично всі сучасні популярні браузери оснащені таким інтерпретатором. Крім JavaScript на HTML-сторінках можливо використовувати інші мови програмування. Наприклад, VBScript або JScript, яка є варіантом JavaScript від фірми Microsoft. Але виконання програм VBScript та JScript гарантовано коректне тільки при перегляді HTML-сторінки за допомогою браузера Microsoft Internet Explorer. Тому в більшості випадків використання JavaScript доцільніше, хоча функціональність програм VBScript та JScript дещо краща.

Досить часто програму JavaScript називають *скриптом* або *сценарієм*. Скрипти виконуються в результаті того, що відбулась деяка подія, пов'язана з HTML-сторінкою. В багатьох випадках виконання вказаних подій ініціюється діями користувача.

Скрипт може бути пов'язаний з HTML-сторінкою двома способами:

- За допомогою парного тегу SCRIPT;
- Як оброблювач події, що стосується конкретного тегу HTML.

Сценарій, вбудований в HTML-сторінку з використанням тегу SCRIPT, має наступний формат:

```
<SCRIPT>  
// Код програми  
</SCRIPT>
```

Все, що розміщується між тегами <SCRIPT> та </SCRIPT>, інтерпретується як код програми на мові JavaScript. Обсяг вказаного коду не обмежений. Інколи скрипти розміщують в середині HTML-коментарію. Це роблять для того, щоб код JavaScript не розглядався старими браузером, які не мають інтерпретатора JavaScript. В цьому випадку сценарій має формат:

```
<SCRIPT>  
<!--  
// Код програми  
-->  
</SCRIPT>
```

Тег SCRIPT має декілька необов'язкових параметрів. Найчастіше використовуються параметри *language* та *src*. Параметр *language* дозволяє визначити мову та версію мови сценарію. Параметр *src* дозволяє задати файл з кодом сценарію.

Також для задання мови сценарію можна використати параметр *type*

```
<script type="text/javascript">
```

Для пояснення використання параметрів тегу SCRIPT розглянемо задачу.

Задача. Необхідно для HTML-сторінки hi. htm створити сценарій на мові JavaScript 1.3 для показу на екрані вікна повідомлення з текстом "Привіт!".

Відзначимо, що для показу на екрані вікна повідомлення можливо використати функцію alert.

Для ілюстрації можливостей пов'язування скриптів з HTML-кодом вирішення задачі реалізуємо двома варіантами.

Варіант 1. Визначення сценарію безпосередньо на HTML-сторінці hi.html

```
<html><head>  
<title>Використання JavaScript</title>  
</head>  
<body>  
<script language="JavaScript1.3">  
alert('hi');  
</script>  
</body></html>
```

Варіант 2. Визначення сценарію в файлі a. js, пов'язаному з HTML-сторінкою hi.html за допомогою параметру src тегу SCRIPT. Код HTML-сторінки hi.html:

```
<html><head>  
<title>Використання JavaScript</title>  
<script language="JavaScript1.3" src="a. js"> </script>  
</head><body>  
</body></html>
```

Програмний код, записаний в файлі a. js:

```
alert('hi');
```

Результат обох варіантів вирішення задачі однаковий. На рис. 1 зображено приклад виконання такого сценарію у браузері Chrome.

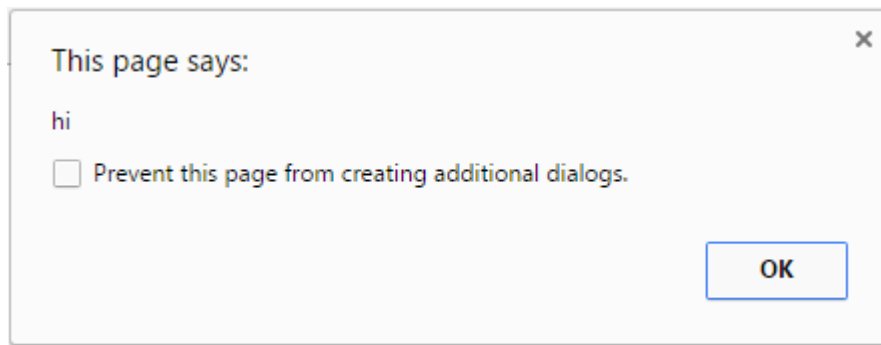


Рис. 1 Виконання сценарію у браузері Chrome.

ХІД РОБОТИ

1. Освоїти основні правила роботи з JavaScript.
2. Переглянути подані зразки скриптів, та спробувати запустити їх у будь-якому браузері.
3. Написати сценарій, який за допомогою команди *alert* виводить інформацію про студента: ПІБ, групу, номер студентського. Зробити це двома способами – з скриптом вбудованим в html файл, та за допомогою окремо підключеного файлу сценарію.

ЗМІСТ ЗВІТУ

1. Короткі теоретичні відомості
2. Код сценаріїв та розмітки згідно з завданням
3. Висновки

ЛІТЕРАТУРА

1. Эд Титтел, Мэри Бурмейстер. HTML 4 для «чайников» = HTML 4 For Dummies. – 5-е изд. – М.: Диалектика, 2006. – 368 с.
2. <http://www.w3.org>
3. Артур Бибек, Бад Смит Создание Web-страниц для "чайников" = Creating Web Pages For Dummies. – 7-е изд. – М.: Диалектика, 2006. – 213 с.

4. Готто Келли, Котлер Эмили Веб-редизайн, 2-е издание. – СПб.: Символ-Плюс, 2006. – 416 с.
5. Флэнаган Д. JavaScript. Подробное руководство = JavaScript. The Definite Guide / Перевод А. Киселева. – 5-е изд. – СПб.: Символ-Плюс, 2008. – 992 с. – ISBN 5-93286-103-7

Лабораторна робота №2 Використання функцій в JavaScript

Мета роботи: Набути навичок з використання функцій на мові JavaScript. Написати на даній мові власну функцію для виконання поставленого завдання згідно з варіантом.

Теоретичні відомості

У більшості програм на мові JavaScript як і у випадках інших мов програмування широко використовуються функції. Функція - у програмуванні - один з видів підпрограми. Особливість, що відрізняє її від іншого виду підпрограм - процедури, полягає в тому, що функція повертає значення, а її виклик може використатися в програмі як вираження.

Підпрограма - частина програми, яка реалізує певний алгоритм і дозволяє звернення до неї з різних частин загальної (головної) програми.

Підпрограма часто використовується для скорочення розмірів програм у тих завданнях, у процесі розв'язання яких необхідно виконати декілька разів однаковий алгоритм при різних значеннях параметрів. Оператори (команди), які реалізують відповідну підпрограму, записують один раз, а в необхідних місцях розміщують оператори передачі управління на цю підпрограму.

З погляду теорії систем, функція в програмуванні - окрема система (підсистема, підпрограма), на вхід якої надходять керуючі впливи у вигляді значень аргументів. На виході системи одержуємо результат виконання програми, що може бути як скалярною величиною, так і векторним значенням. По ходу виконання функції можуть виконуватися також деякі зміни в керованій системі, причому як оборотні, так і необоротні.

У деяких мовах програмування (наприклад, у Паскалі) функції існують поряд із процедурами (підпрограмами, що не повертають значення), в інші, наприклад, в С, є єдиним реалізованим видом підпрограми (тобто всі підпрограми є функціями й можуть повертати значення).

Побічним ефектом функції називається будь-яка зміна функцією стану програмного середовища, крім повернення результату (зміна значень глобальних змінних, виділення й звільнення пам'яті, ввід-висновки і так далі).

Теоретично найбільш правильним є використання функцій, що не мають побічного ефекту (тобто таких, у результаті виклику яких повертається обчислене значення, і тільки), хоча на практиці доводиться використати функції з побічним ефектом, хоча б для забезпечення вводу-висновку й відображення результатів роботи програми.

Для прикладу використання функцій у мові JavaScript розглянемо скрипт, що друкує деякий текст три рази поспіль:

```
<html>
<script language="JavaScript">
document.write("Ласкаво просимо на мою сторінку!<br>");
document.write("Це JavaScript!<br>");
document.write("Ласкаво просимо на мою сторінку!<br>");
document.write("Це JavaScript!<br>");
document.write("Ласкаво просимо на мою сторінку!<br>");
document.write("Це JavaScript!<br>");
</script>
</html>
```

Такий скрипт напише

«Ласкаво просимо на мою сторінку!

Це JavaScript!»

тричі. Як бачимо поставлена задача вирішена за допомогою того, що відповідна частина коду була дубльована три рази. Такий підхід не є ефективним. Наступний скрипт виконує ту ж саму задачу з використанням функції.

```
<html>
```

```
<script language="JavaScript">
```

```
function myFunction()
```

```
{
```

```
document.write("Ласкаво просимо на мою сторінку!<br>");
```

```
document.write("Это JavaScript!<br>");
```

```
}
```

```
myFunction();
```

```
myFunction();
```

```
myFunction();
```

```
</script>
```

```
</html>
```

І хоча кількість рядків у даному випадку не змінилась структура програми стала більш зрозумілою, читабельною, та придатною до подальшої модифікації. У даному програмному коді ми використали функцію, що містить наступні строки:

```
function myFunction()
```


{

```
document.write("Ласкаво просимо на мою сторінку!<br>");
```

```
document.write("Це JavaScript!<br>");
```

}

Для визначення функції використовується ключове слово `function`. `myFunction` у даному випадку ім'я функції. У круглих дужках може бути вказаний ряд параметрів. Усі команди скрипта, що знаходяться між фігурними скобками – `{}` – належать функції `myFunction()` та виконуються при її виклику. Це означає, що обидві команди `document.write()` пов'язані між собою та можуть бути виконані при виклику даної функції.

Практичне завдання

Завдання: Розробити Інтернет сторінку на якій знаходиться лише одна кнопка. На подію натиснення “`onClick`” призначити виконання функції розрахунку арифметичної операції вибраної за номером студента в списку (значення змінних задати самостійно у функції):

№ п/п	F	№ п/п	
1	$F = x + 2y$	14	$F = \frac{1}{x + y + z}$
2	$F = 2x - 3y^2$	15	$F = \frac{x - y}{2xz}$
3	$F = \frac{1 - x^2}{y^3}$	16	$F = 21xy - xz + xyz$
4	$F = 2x + 3y - z$	17	$F = x^3 + x^2 + x + xy^2$

5	$F = 3xy - y + 2x$	18	$F = x + y^2 + z^{0.5}$
6	$F = \frac{x}{y} + \frac{y^2}{1 - xy}$	19	$F = x^{-0.5}$
7	$F = x - xy + y^2$	20	$F = x^6 + x^5y + 0.5z + x^2$
8	$F = x + \sqrt{y}$	21	$F = \frac{xz}{y}$
9	$F = xy + xz - yz$	22	$F = \frac{\sqrt{x}}{y + z}$
10	$F = 2x + y - \frac{z}{xy}$	23	$F = \frac{x - y^{0.5}}{xz}$
11	$F = 1 - x - y - z$	24	$F = -3xz + \sqrt{x^3}$
12	$F = \left(1 - \frac{x}{y}\right)^2 + z$	25	$F = x^{y^z} + 2xz$
13	$F = x^y + y^z + z^x$	26	$F = \frac{\sqrt{y}}{\sqrt[3]{xz}}$

Приклад програми:

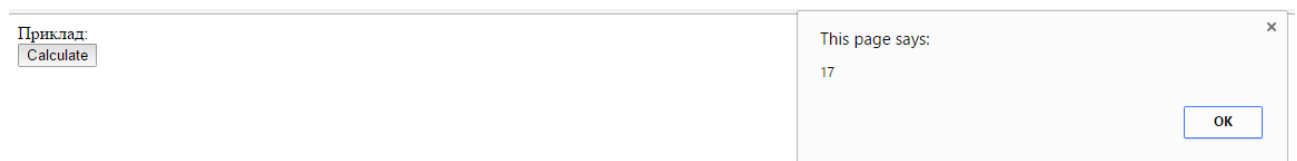
```

<html>
<head>
<script language="JavaScript">
function calculation() {
var x= 12;
var y= 5;
var result= x + y;
alert(result);
}
</script>
</head>

```

```
<body>
<form>
<input type="button" value="Calculate" onClick="calculation()">
</form>
</body>
</html>
```

Вигляд сторінки та результат виконання:



ХІД РОБОТИ

1. Освоїти основні правила роботи з функціями JavaScript.
2. Переглянути подані зразки скриптів, та спробувати запустити їх у будь-якому броузері.
3. Написати сценарій, який після натискання кнопки на сторінці виводитиме результат обчислення математичної функції, вирахований за допомогою використання функції JavaScript.

ЗМІСТ ЗВІТУ

1. Короткі теоретичні відомості
2. Код сценаріїв та розмітки згідно з завданням
3. Висновки

ЛІТЕРАТУРА

1. Эд Титтел, Мэри Бурмейстер. HTML 4 для «чайников» = HTML 4 For Dummies. – 5-е изд. – М.: Диалектика, 2006. – 368 с.
2. <http://www.w3.org>

3. Артур Бибек, Бад Смит Создание Web-страниц для "чайников" = Creating Web Pages For Dummies. – 7-е изд. – М.: Диалектика, 2006. – 213 с.
4. Гото Келли, Котлер Эмили Веб-редизайн, 2-е издание. – СПб.: Символ-Плюс, 2006. – 416 с.
5. Флэнаган Д. JavaScript. Подробное руководство = JavaScript. The Definite Guide / Перевод А. Киселева. – 5-е изд. – СПб.: Символ-Плюс, 2008. – 992 с. – ISBN 5-93286-103-7

Лабораторна робота №3 Події в JavaScript

Мета роботи: Набути навичок з використання функцій на мові JavaScript. Написати на даній мові власну функцію для виконання поставленого завдання згідно з варіантом.

Теоретичні відомості

Практично всі JavaScript-додатки виконують ті чи інші дії, відгукуючись на різні події. Подія - це сигнал від браузера про те, що щось сталося.

Є безліч найрізноманітніших подій.

- DOM-події, які ініціюються елементами DOM. Наприклад, подія click відбувається при кліці на елементі, а подія mouseover - коли курсор миші з'являється над елементом,
- Події вікна. Наприклад подія resize - при зміні розміру вікна браузера,
- Інші події, наприклад load. Вони використовуються, скажімо, в технології AJAX.

Саме DOM-події пов'язують події, які у документі, з кодом JavaScript, тим самим забезпечуючи динамічний веб-інтерфейс.

Призначення обробників подій в JavaScript

Події можна призначити обробник, тобто функцію, яка спрацює, як тільки подія відбулася. Саме завдяки обробникам JavaScript-код може реагувати на дії відвідувача.

Потрібно відразу відзначити, що JavaScript - однопотокова мова, тому обробники завжди виконуватимуться послідовно і в загальному потоці. Це означає, що при установці обробників двох подій, які виникають на елементі одночасно, наприклад mouseover (миша з'явилася над елементом) і mousemove (миша рухається над елементом), їх обробники будуть виконані послідовно.

Є кілька способів призначити події обробник. Зараз ми їх розглянемо, починаючи від самого простого.

Використання атрибуту HTML

Обробник події можна вказати у вигляді inline-запису, прямо в атрибуті on<подія>.

Наприклад, для обробки події click на кнопці input, можна призначити обробник onclick ось так:

```
<input value="Натисни мене" onclick="alert('Дякую!');" type="button">
```

Як ми пам'ятаємо, атрибут HTML-тега не чутливий до регістру, тому ONCLICK буде працювати так само, як onClick або onCLICK ... Але, як правило, атрибути пишуть в нижньому регістрі: onclick.

Такий спосіб призначення обробників дуже зручний - він наочний і простий, тому часто використовується в рішенні простих завдань.

У цього способу установки обробника є й мінуси. Як тільки обробник починає займати більше одного рядка - читабельність різко падає.

Втім, скільки-небудь складні обробники в HTML ніхто не пише. Замість цього краще встановлювати обробники з JavaScript способами, які будуть представлені нижче.

Використання властивості DOM-об'єкта

Найближчий родич описаного вище способу - установка функції-обробника через властивість on<подія> відповідного елемента. Цей спосіб теж буде працювати в будь-якому браузері з підтримкою JavaScript.

Для цього потрібно:

- отримати елемент
- призначити обробник властивості on<ім'я>

Ось приклад установки обробника події click на елемент з id = "myElement":

```
document.getElementById('myElement').onclick = function() {  
  
    alert('Дякую')  
  
}
```

Всі виклики типу getElementById повинні запускатися після опису відповідного HTML-вузла, а краще - після закінчення завантаження сторінки. Інакше вузол просто не буде знайдений.

Описане встановлення обробника через властивість - дуже популярний і простий спосіб. У нього є один недолік: на елемент можна повісити тільки один обробник потрібної події.

Декілька найпопулярніших обробників подій:

Атрибут HTML	Умова виникнення події
onBlur	Втрата фокуса елементом форми
onChange	Зміна поля вводу або області тексту, або вибір нового елемента списку
onClick	Клік миші на елементі форми або гіперпосилання'

onFocus	Отримання фокуса вводу елементом форми
onLoad	Завершення завантаження документа
onMouseOver	Зміщення вказівника миші на гіперпосилання
onMouseOut	Зміщення вказівника миші не на гіперпосилання
onSelect	Виділення тексту в полі вводу або області тексту
onSubmit	Передача даних форми
onUnload	Вивантаження поточного документу та початок завантаження нового

Практичне завдання

Завдання: Завдання: створити html документ, в якому обробити події згідно номеру варіанта.

№ п/п	Події	№ п/п	Події
1	onChange, onClick	15	onFocus, onMouseOut
2	onClick, onLoad	16	onBlur, onClick
3	onBlur, onMouseOver	17	onClick, onMouseOver
4	onFocus, onClick	18	onChange, onSelect
5	onLoad, onMouseOut	19	onFocus, onChange
6	onBlur, onChange	20	onLoad, onMouseOut
7	onChange, onMouseOver	21	onClick, onFocus
8	onFocus, onSelect	22	onChange, onLoad
9	onClick, onChange	23	onBlur, onSelect
10	onFocus, onSelect	24	onFocus, onMouseOver
11	onClick, onMouseOut	25	onChange, onFocus
12	onLoad, onMouseOver	26	onLoad, onMouseOut
13	onBlur, onSelect	27	onClick, onSelect
14	onChange, onMouseOut	28	onBlur, onMouseOver

Приклад:

```

<html>
<head>
<script>
function fun(a)
{
main_form.t_res.value = a;
}

```



```

</script>
</head>
<body onLoad = "fun('Відбулось завантаження сторінки');">
<form id = "main_form">
<input type="text" size = "100" value="Calculate" onClick="fun('Відбувся клік на текстовому
елементі форми');" onBlur = "fun('Текстовий елемент форми втратив фокус');" onChange =
"fun('Зміст текстового поля форми змінений');" onFocus = "fun('Текстовий елемент форми
отримав фокус');" onSelect = "fun('В текстовому полі форми виділений текст');" ><br>
<input type = "text" size = "100" id = "t_res" onMouseOver = "fun('Курсор потрапив на
гіперпосилання');" onMouseOut = "fun('Курсор потрапив не на гіперпосилання');" ><br>
<a href = "d:\"" >Посилання</a>
</form>
</body>
</html>

```

Вигляд

сторінки:

Calculate
Курсор потрапив не на гіперпосилання
Посилання

ХІД РОБОТИ

1. Освоїти основні правила роботи з подіями JavaScript.
2. Переглянути подані зразки скриптів, та спробувати запустити їх у будь-якому браузері.
3. Написати сценарій, який оброблятиме задані події.

ЗМІСТ ЗВІТУ

1. Короткі теоретичні відомості
2. Код сценаріїв та розмітки згідно з завданням
3. Висновки

ЛІТЕРАТУРА

1. Эд Титтел, Мэри Бурмейстер. HTML 4 для «чайников» = HTML 4 For Dummies. – 5-е изд. – М.: Диалектика, 2006. – 368 с.
2. <http://www.w3.org>
3. Артур Бибек, Бад Смит Создание Web-страниц для "чайников" = Creating Web Pages For Dummies. – 7-е изд. – М.: Диалектика, 2006. – 213 с.
4. Гото Келли, Котлер Эмили Веб-редизайн, 2-е издание. – СПб.: Символ-Плюс, 2006. – 416 с.
5. Флэнаган Д. JavaScript. Подробное руководство = JavaScript. The Definite Guide / Перевод А. Киселева. – 5-е изд. – СПб.: Символ-Плюс, 2008. – 992 с. – ISBN 5-93286-103-7

Лабораторна робота №4 Умовні вирази, оператори присвоювання та порівняння, логічні операції, коментар у JavaScript

Мета роботи: Набути навичок з використання умовних виразів, операторів присвоювання та порівняння, логічних операцій, коментарів у JavaScript. Написати на даній мові власну функцію для виконання поставленого завдання згідно з варіантом.

Теоретичні відомості

При розробці функціональної бази Інтернет сторінок з використанням мови JavaScript одним з необхідних елементів є умовні оператори. Виконання складних математичних операцій вимагає паралельної перевірки деяких умов. Структура умовного оператора в JavaScript має наступний вигляд:

if(<логічний_вираз>) <оператор_1> else <оператор_2>

У даному випадку <логічний_вираз> представляє собою повний або скорочений варіант певного виразу. При умові коли <логічний_вираз> приймає значення *true* виконується <оператор_1>, у іншому разі, тобто коли його значення дорівнює *false* <оператор_2>. При побудові логічних виразів здебільшого застосовуються наступні операції порівняння:

№ з/п	Позначення	Зміст
1	==	дорівнює
2	!=	не дорівнює
3	>	більше
4	>=	більше або дорівнює
5	<	менше
6	<=	менше або дорівнює

Слід зазначити, що у більшості випадках для скорочення програмного коду та підвищення рівня його читабельності використовуються наступні спрощення при записі логічних виразів:

<i>a == 0</i>	<i>!a</i>
<i>a != 0</i>	<i>a</i>
<i>str == ""</i>	<i>!str</i>
<i>str != ""</i>	<i>str</i>

Можливі випадки коли при побудові логічних виразів необхідно врахувати не одну а кілька умов. Мова Java Script передбачає використання спеціальних символів побудови складних умовних операторів:

Позначення	Оператор
& &	і
	або

Структурний елемент *else* умовного оператора *if* у більшості випадках використовується для оптимізації програмного коду та удосконалення його читабельності. Слід зазначити, що у випадках коли варіанти перебору умовного оператора *if* є взаємовиключними, використання оператора *else* дозволяє скоротити об'єм перевірок і як результат пришвидшити роботу програми. У випадку коли варіанти перебору не доповнюють один одного використання структурного елементу *else* може спровокувати виникнення помилки. Нижче наведені два фрагменти коду, що демонструють не тільки оптимізаційні властивості умовних операторів, але й варіанти їх використання:

Взаємовиключаючі умови	Взаємодоповнюючі умови
<pre>if(a==1) b=a; else if(a==2) b=a*a; else if(a==3) b=a*a*a; else if(a==4) b=1/a; else if(a==5) b=1/a*a; else</pre>	<pre>if(str.length) flag=1; if(((str[2]=="a") (str[2]=="k"))&&(str[0]==" ")) flag=0; if(str.length>=5) flag=str.length if(str=="twenty") {number= 20; flag=0;}</pre>

Практичне завдання

Завдання: Створити html документ для реалізації розгалуженого обчислювального процесу згідно варіанту завдання:

№ з/п	Варіант завдання	№ з/п	Варіант завдання
1	$F = \begin{cases} x + y, xy > 0 \\ x, x^2 > 100 \\ x^3 + y, \text{інакше} \end{cases}$	15	$F = \begin{cases} x + y, xy \neq 0 \\ x, x^3 > 100 \\ x^3 + y, \text{інакше} \end{cases}$
2	$F = \begin{cases} x + y, (x > 0) \text{та} (y > 0) \\ x^2 + y^2, (x < 3) \text{та} (y < 4) \\ x^3, \text{інакше} \end{cases}$	16	$F = \begin{cases} 2x + y, (x > 0) \text{та} (y > 0) \\ x^2 + 2y^2, (x < 3) \text{та} (y > 4) \\ x^3, \text{інакше} \end{cases}$

3	$F = \begin{cases} x^2 + y^2, (x < 10) \text{ або } (y > 3) \\ x - y + x^2, x > 2 \\ x^3 + 3xy, \text{інакше} \end{cases}$	17	$F = \begin{cases} 2x^2 - 3y^2, (x < 10) \text{ або } (y \neq 3) \\ 2x - 3y/x + x^2, x \neq 2 \\ x^3 + 3xy, \text{інакше} \end{cases}$
4	$F = \begin{cases} x/y + xy^2, (x > 0) \text{ та } (y > 0) \\ y/x, (-x + y)/2 > 0 \\ x^2, \text{інакше} \end{cases}$	18	$F = \begin{cases} 2x/y + xy^2, (x > 0) \text{ та } (y \neq 0) \\ y/2x, (2x + y)/2 \neq 0 \\ x^2, \text{інакше} \end{cases}$
5	$F = \begin{cases} x - y^2, x - y > 2 \\ x/y, x > 0 \\ x^3 - y^2, \text{інакше} \end{cases}$	19	$F = \begin{cases} x - 2y^2, x - 3y \neq 2 \\ x - y/y, x > 0 \\ 2x^2 - 3y, \text{інакше} \end{cases}$
6	$F = \begin{cases} x/2y, xy < -100 \\ x^2 - xy, xy > 20 \\ x^3 + 2, \text{інакше} \end{cases}$	20	$F = \begin{cases} 2x/y, xy \neq -100 \\ x^3 - 2xy, xy \neq 20 \\ x^2 + 4, \text{інакше} \end{cases}$
7	$F = \begin{cases} 2xy - 3, x/y > 0 \\ xy/3, (x > 0) \text{ та } (y < 3) \\ 2x - 3y, \text{інакше} \end{cases}$	21	$F = \begin{cases} xy - 1, x/y > 0 \\ 2xy/2, (x \neq 0) \text{ та } (y \neq 3) \\ x - y, \text{інакше} \end{cases}$
8	$F = \begin{cases} 2x + 3y, x < 3y \\ xy + x/3y, (y^2 > 2xy) \text{ або } (y > 0) \\ x^2 + y, \text{інакше} \end{cases}$	22	$F = \begin{cases} 10x + y, x \neq y \\ 2xy + 4x/y, (y^2 \neq 2xy) \text{ або } (y < 0) \\ x + y^2, \text{інакше} \end{cases}$
9	$F = \begin{cases} x^3 - \sqrt{x}, x^2 > 3 \\ xy, (x > 0) \text{ та } (y < 0) \\ x + y + xy - x/y, \text{інакше} \end{cases}$	23	$F = \begin{cases} x^3, x^3 \neq 3 \\ xy, (x \neq 0) \text{ та } (y < 0) \\ 2x + xy - x/2y, \text{інакше} \end{cases}$
10	$F = \begin{cases} 3xy - x, xy > 0 \\ 2x - 3y/x, (7x > 2y) \text{ або } (y > 2) \\ x - y, \text{інакше} \end{cases}$	24	$F = \begin{cases} xy - x, 2x/y \neq 0 \\ (x - 2y)/x, (14x \neq y) \text{ або } (2y < -4) \\ x - y, \text{інакше} \end{cases}$

11	$F = \begin{cases} 3xy, x > 0 \\ 2x/y, (x < 0) \text{ та } (y > 3/2) \\ x^2 + y^3, \text{інакше} \end{cases}$	25	$F = \begin{cases} x/y, x \neq 0 \\ 2x/y, (x \neq 0) \text{ або } (y > 3/2) \\ -x^3 + 2y^2, \text{інакше} \end{cases}$
12	$F = \begin{cases} x - y, xy > 0 \\ x^2 - y/x, (y < 0) \text{ та } (x > 3y) \\ 3xy - x^2y^2, \text{інакше} \end{cases}$	26	$F = \begin{cases} 2x - y, 2xy \neq 0 \\ x^2 + 2y/x, (y \neq 0) \text{ або } (x \neq 3y) \\ x/y - xy, \text{інакше} \end{cases}$
13	$F = \begin{cases} 2\sqrt{x} \cdot y, (y > 3) \text{ та } (x > 0) \\ x/y^2, (yx^2/2 > 3x) \text{ або } (x^2 > 2y) \\ x^2/y, \text{інакше} \end{cases}$	27	$F = \begin{cases} (3/5)\sqrt{x} \cdot y, (y \neq 5) \text{ та } (x \neq 0) \\ x/y^2, (yx^2 \neq 3x) \text{ або } (x^3 \neq 5xy) \\ x/y^2, \text{інакше} \end{cases}$
14	$F = \begin{cases} -3x + y^3, (x > 0) \text{ та } (y < 0) \\ x + y - x^2y, xy - x < 3 \\ x^2 + y^2, \text{інакше} \end{cases}$	28	$F = \begin{cases} x + y^2, (x \neq 0) \text{ та } (y \neq 0) \\ x - y + x^3y^2, xy - x \geq 3 \\ x^2 - y^2, \text{інакше} \end{cases}$

Приклад:

```

<html>
<head>
<script>
function fun(){
var a, b, c, d;
a = parseInt(main_form.t_a.value);
b = parseInt(main_form.t_b.value);
c = parseInt(main_form.t_c.value);
if(a*b*c>0) d=a*b-c; else if(a*b*c<0) d=a-b*c; else
d=a*c-Math.sqrt(b);
main_form.t_d.value = "" + d;}
</script>
</head>
<body>
<form id = "main_form">
a<input type="text" id = "t_a" value="10"><br>
b<input type="text" id = "t_b" value="-2"><br>
c<input type="text" id = "t_c" value="3"><br>
d<input type="text" id = "t_d" value=""><br>
<input type = "button" onClick = "fun();" value = "Розрахувати">
</form>
</body>
</html>

```

Результат виконання:

a	10
b	-2
c	3
d	16
<input type="button" value="Розрахувати"/>	

ХІД РОБОТИ

1. Освоїти основні правила роботи з умовними виразами, операторами присвоювання та порівняння, логічними операціями, коментарями JavaScript.
2. Переглянути подані зразки скриптів, та спробувати запустити їх у будь-якому браузері.
3. Написати сценарій, який обчислюватиме задану функцію.

ЗМІСТ ЗВІТУ

1. Короткі теоретичні відомості
2. Код сценаріїв та розмітки згідно з завданням
3. Висновки

ЛІТЕРАТУРА

1. Эд Титтел, Мэри Бурмейстер. HTML 4 для «чайников» = HTML 4 For Dummies. – 5-е изд. – М.: Диалектика, 2006. – 368 с.
2. <http://www.w3.org>
3. Артур Бибек, Бад Смит Создание Web-страниц для "чайников" = Creating Web Pages For Dummies. – 7-е изд. – М.: Диалектика, 2006. – 213 с.
4. Гото Келли, Котлер Эмили Веб-редизайн, 2-е издание. – СПб.: Символ-Плюс, 2006. – 416 с.
5. Флэнаган Д. JavaScript. Подробное руководство = JavaScript. The Definite Guide / Перевод А. Киселева. – 5-е изд. – СПб.: Символ-Плюс, 2008. – 992 с. – ISBN 5-93286-103-7

Лабораторна робота №5 Створення динамічних інтерфейсів засобами JavaScript

Мета роботи: Набути навичок з використання динамічних інтерфейсів у JavaScript. Написати на даній мові власну функцію для виконання поставленого завдання згідно з варіантом.

Теоретичні відомості

У більшості випадках інформаційні html сайти працюють в online режимі безперервно відображаючи інформацію, що надходить з певних баз даних, архівів та інших джерел. З самого початку розробник не знає яким чином буде підбиратись інформація, скільки інформаційних стрічок буде виведено на головну сторінку тощо. Саме тому використання статичних об'єктів незручно при створенні інтерактивних сайтів. Динамічні інтерфейси є одним з визнаних інструментів розробки гнучкої платформи адаптованої під певну діяльність. Сутність методу динамічних інтерфейсів полягає в розміщенні на сторінці спеціальних елементів `<div>`. В процесі роботи коли кількість та сутність інформації, що має у певний час відображатись на сторінці, визначена, html код генерується автоматично самою програмою. Після чого він інкапсулюється в `<div>`. Інкапсуляція html коду в div відбувається з використанням функції `innerHTML`. Таким чином можна зчитуючи інформацію з бази даних створювати автоматично бланки анкетного опитування, при цьому його редагування стає напрочуд простим та стійким до помилок.

Практичне завдання

Завдання: розробити html документ в якому існуватиме два текстових елемента “кількість рядків” та “кількість стовпців”. Згідно із їх значенням створити таблицю текстових елементів (див. приклад) кожний елемент якого заповнити відповідно індивідуального варіанту. Для варіантів 1, 6, 11, 16, 21, 26 підрахувати суму елементів отриманого двовимірного масиву. Для варіантів 2, 7, 12, 17, 22,

27 підрахувати суму елементів другого рядка. Для варіантів 3, 8, 13, 18, 23, 28 підрахувати суму елементів третього стовпця. Для варіантів 4, 9, 14, 19, 24 підрахувати суму елементів порядковий номер строки та стовпця яких співпадають. Для варіантів 5, 10, 15, 20, 25 підрахувати добуток елементів останнього стовпця розробити html документ в якому існуватиме два текстових елемента “кількість рядків” та “кількість стовпців”. Згідно із їх значенням створити таблицю текстових елементів (див. приклад) кожний елемент якого заповнити відповідно індивідуального варіанту. Для варіантів порядковий номер якого кратний. Для варіантів 1, 6, 11, 16, 21, 26 підрахувати суму елементів отриманого двовимірного масиву. Для варіантів 2, 7, 12, 17, 22, 27 підрахувати суму елементів другого рядка. Для варіантів 3, 8, 13, 18, 23, 28 підрахувати суму елементів третього стовпця. Для варіантів 4, 9, 14, 19, 24 підрахувати суму елементів порядковий номер строки та стовпця яких співпадають. Для варіантів 5, 10, 15, 20, 25 підрахувати добуток елементів останнього стовпця.

№ з/п	Варіант завдання	№ з/п	Варіант завдання
1	$A_{ij} = 2i + j$	15	$A_{ij} = 7j + 2i$
2	$A_{ij} = 2i / j$	16	$A_{ij} = 2j - i$
3	$A_{ij} = 11i + j$	17	$A_{ij} = -5j + 3i$
4	$A_{ij} = 2i + 13j$	18	$A_{ij} = 3i - 2j$
5	$A_{ij} = 3i + 2j$	19	$A_{ij} = 11j - i$
6	$A_{ij} = 2i / 5j$	20	$A_{ij} = 2j / 5i$
7	$A_{ij} = 1/(2 + j)$	21	$A_{ij} = 2j + i$
8	$A_{ij} = 11j + i$	22	$A_{ij} = -7j / 3i$
9	$A_{ij} = 5i + 3j$	23	$A_{ij} = 11i - j$
10	$A_{ij} = 2i - j$	24	$A_{ij} = 2j / i$

11	$A_{ij} = -2i / 7j$	25	$A_{ij} = 3j + 2i$
12	$A_{ij} = 3i + j$	26	$A_{ij} = i - j$
13	$A_{ij} = 3j - 2i$	27	$A_{ij} = 5j + 3i$
14	$A_{ij} = -5i + 3j$	28	$A_{ij} = (3 - i)/(j + 5)$

Приклад:

```

<html>
<head>
<script>
var str, stb;
function fun()
{
str = parseInt(main_form.t_str.value);
stb = parseInt(main_form.t_stb.value);
var res_str = "<table>\n";
for(var i=1;i<=str;i++)
{
res_str+="<tr>\n";
for(var j=1;j<=stb;j++)
{
res_str += "<td>";
res_str += "<input type = \"text\" id = \"_\" + i + \"_\" + j + \"\" value = \"" + i + "\" + j + "\">";
res_str += "</td>\n";
}
res_str+="</tr>\n";
}
res_str += "</table>";
main_div.innerHTML = res_str;
}
function fun_build()
{
var res_str = "";
var str_report = "";

```

```

for(var i=1;i<=str;i++)
{
var sum = 0;
for(var j=1;j<=stb;j++)
{
res_str = "sum += parseInt(main_form._" + i + "_" + j + ".value);";
eval(res_str);
}
str_report += "Сумма " + i + " рядка = " + sum + ";\n"
}
alert(str_report);
}
</script>
</head>
<body>
<form id = "main_form">
<table>
<tr>
<td>Кількість рядків</td>
<td> <input type="text" id = "t_str" value="5"></td>
</tr>
<tr>
<td> Кількість стовбців</td>
<td><input type="text" id = "t_stb" value="5"></td>
</tr>
<tr>
<td> <input type = "button" onClick = "fun();" value = "Побудувати матрицю"><br> <td>
</tr>
<tr>
<td> <input type = "button" onClick = "fun_build();" value = "Розрахувати"> <td>
</tr>
</table>
<div id = "main_div"></div>
</form>
</body>
</html>

```

Результат виконання:

Кількість рядків	5		
Кількість стовбців	5		
<input type="button" value="Побудувати матрицю"/>			
<input type="button" value="Розрахувати"/>			
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54

This page says:

Сумма 1 рядка = 65;
Сумма 2 рядка = 115;
Сумма 3 рядка = 165;
Сумма 4 рядка = 215;
Сумма 5 рядка = 265;

☐ Prevent this page from creating additional dialogs.

OK

ХІД РОБОТИ

1. Освоїти основні правила використання динамічних інтерфейсів у JavaScript
2. Переглянути подані зразки скриптів, та спробувати запустити їх у будь-якому браузері.
3. Написати сценарій, який виконуватиме завдання згідно з варіантом.

ЗМІСТ ЗВІТУ

1. Короткі теоретичні відомості
2. Код сценаріїв та розмітки згідно з завданням
3. Висновки

ЛІТЕРАТУРА

1. Эд Титтел, Мэри Бурмейстер. HTML 4 для «чайников» = HTML 4 For Dummies. – 5-е изд. – М.: Диалектика, 2006. – 368 с.
2. <http://www.w3.org>
3. Артур Бибек, Бад Смит Создание Web-страниц для "чайников" = Creating Web Pages For Dummies. – 7-е изд. – М.: Диалектика, 2006. – 213 с.

4. Гото Келли, Котлер Эмили Веб-редизайн, 2-е издание. – СПб.: Символ-Плюс, 2006. – 416 с.
5. Флэнаган Д. JavaScript. Подробное руководство = JavaScript. The Definite Guide / Перевод А. Киселева. – 5-е изд. – СПб.: Символ-Плюс, 2008. – 992 с. – ISBN 5-93286-103-7

Лабораторна робота № 6

Робота із календарною датою засобами мови JavaScript

Мета: отримати теоретичні та практичні навички роботи з датою та часом за допомогою мови Javascript

Теоретичні відомості

Змінні типу дати та часу створюються за допомогою оператора `new`. Так для створення нової змінної типу дати та присвоєння їй значення поточної дати використовується конструкція “`змінна = new Date()`”. Для створення змінної типу дати з певним наперед заданим значенням використовується наступна конструкція “`new Date (year, month, date, hour, minute, second)`”, де `year`—рік, `month`—місяць, `date`—число, `hour`—години, `minute`—хвилини, `second`—секунди відповідно.

Взагалі змінна типу дати є цілочисельною змінною, яка містить кількість мілесекунд, від опівночі першого січня 1970-го року за універсальним часом (UTC).

Функції для роботи з датами поділяються на такі, що зчитують певні параметри та встановлюють параметри. Функції, які зчитують параметри починаються на `get`, які встановлюють параметри – на `set`.

- x. `get Year ()`** – повертає рік
- x. `get Month ()`** – повертає номер місяця
- x. `get Date ()`** – повертає день місяця 1, 2,...
- x. `get Day ()`** – повертає день тижня
- x. `get Hours ()`** – повертає години
- x. `get Minutes ()`** – повертає хвилини

x. get Seconds () – повертає секунди

x. get Time () – повертає кількість секунд починаючи з 1.01.1970

x. set Year (a) – встановлює рік

x. set Month (a) – встановлює місяць 0-січень

x. set Date (a) – встановлює число

x. set Hours (a) – встановлює години 0-23

x. set Minutes (a) – встановлює хвилини

x. set Seconds (a) – встановлює секунди

(a – ціле)

Спеціальні функції

x. tolocalString () – повертає рядок, що представляє дату та час

“mm/dd/yy hh:mm:ss”

x. toString () – видруковує дату та час з словесним представленням місяця та дня тижня.

Date (year, month, date, hour, minute, second) – перетворює дату у число мілісекунд з 1.01.1970р.

У нижченаведеному прикладі виводиться поточний час при відкриванні Web сторінки.

<HTML>

<SCRIPT LANGUAGE = JavaScript>

var dateNow = new Date()

document.write(“Поточний час”)

**document.write(“
”)**

document.write(dateNow.toLocaleString())

</SCRIPT>

</HTML>

У прикладі, наведеному вище, відрукований час відповідає часу відкривання Web сторінки, і не змінюється. Для того щоб час динамічно змінювався, потрібно його циклічно зчитувати та відруковувати з інтервалом у одну секунду. Наступний приклад демонструє виведення поточного часу у рядок стану браузера.

```
<html>
<head>
<title>Годинник в рядку стану</title>

<script language="JavaScript">
function clock_status()
{
    window.setTimeout("clock_status()",1);
    today=new Date();
    self.status=today.toString();
}
</script>
</head>
<body background="ffffff" onLoad="clock_status()">
</body>
</html>
```

Завдання

1. Web сторінка містить кнопку “Старт” та поле вводу. З клавіатури у поле вводиться кількість секунд затримки та натискається кнопка. Через відповідну кількість секунд має з’явитися напис “Інтервал завершено”.
2. При відкриванні Web сторінки, в залежності від пори дня має з’явитися відповідне привітання: “Добрий ранок”, “Дорий день” та “Добрий вечір”.

3. При відкриванні Web сторінки має з'явитися інформація про те, яка йде пара і у якій аудиторії у групи заданої варіантом студента. Також мають враховуватися перерви.

Варіанти:

- 1- КН-10
- 2- КН-11
- 3- КН-12
- 4- КН-13
- 5- КН-14
- 6- КН-15
- 7- КН-16
- 8- КН-20
- 9- КН-21
- 10- КН-22
- 11- КН-23
- 12- КН-24
- 13- КН-25
- 14- КН-26
- 15- КН-30
- 16- КН-31
- 17- КН-32
- 18- КН-33
- 19- КН-34
- 20- КН-35
- 21- КН-36
- 22- КН-40
- 23- КН-41
- 24- КН-42
- 25- КН-43
- 26- КН-44
- 27- КН-45

28- КН-46

29- ПИ-10

30- ПИ-11

ЛАБОРАТОРНА РОБОТА №7-8

Розробка програм асинхронного обміну даними у Web-середовищі засобами бібліотеки JQuery. Ajax.

Метою лабораторної роботи є вивчення основ асинхронного обміну даними мовою JavaScript за допомогою технології Ajax та бібліотеки JQuery при реалізації практичних задач Web-програмування.

Завдання на лабораторну роботу:

1. Ознайомитись із синтаксисом та базовими можливостями бібліотеки JQuery.
2. Набути практичних навичок маніпулювання вмістом Web-сторінок.
3. Вивчити способи реалізації інтерактивного інтерфейсу користувача за допомогою засобів JQuery Ajax.

Основні теоретичні відомості

Бібліотека JQuery як інструмент розробки інтерактивних Web-додатків

Активний розвиток застосування різноманітних Web-додатків, зокрема, соціальних мереж та засобів електронної комерції в Internet, зумовив необхідність створення інтерактивних інтерфейсів користувача, які за функціональними можливостями наближуються до традиційних віконних додатків. Місце програмного інструментарію для вирішення цієї задачі за останні 5-7 років зайняли різноманітні бібліотеки на основі вбудованої в Web-браузер мови Javascript. Серед таких засобів найбільш поширеною та вдалою виявилась бібліотека JQuery. Вона поєднує в собі лаконічний синтаксис, надзвичайно розвинуті засоби пошуку елементів Web-сторінки на основі CSS та XML-орієнтованої спеціалізованої мови XPath, функції маніпуляції XHTML DOM, анімаційні ефекти, а також ефективний інструментарій асинхронного обміну інформацією з Web-сервером Ajax.

Базовий синтаксис команди JQuery

Оскільки бібліотека JQuery написана мовою Javascript, то застосовувати код можна всюди, де допустимий код Javascript: в оброблювачах подій, викликаючи функції тощо. Спрощено запис команди JQuery виглядає наступним чином:

\$(selector).action1(args1).action2(args2) ...

або

\$.action(args)

де знак *\$* означає скорочену назву об'єкта JQuery. Цей знак можна замінити на виклик: JQuery(selector);

selector визначає один або групу елементів Web-документа, об'єднаних за певним критерієм (назвою тега, ідентифікатору, CSS-класу тощо);

action, *action1*, *action2* – визначають функцію, що треба застосувати для відібраних селектором елементів (встановити стиль CSS, заповнити/стерти вміст елемента, додати/вилучити оброблювач деякої події тощо);

arg, *arg1*, *arg2* – необов'язкові аргументи функцій, що застосовуються до елементів (новий вміст елемента, його атрибути тощо).

Якщо вказано декілька дій (*action*) до відібраних селекторів, то вони виконуються послідовно, наприклад:

\$("p").html("abcd").css("color", "green")

В цьому прикладі для усіх параграфів буде внесено текст —*abcd* та встановлено колір тексту в зелений.

Селектори JQuery

Бібліотека JQuery надає широкий спектр можливостей щодо пошуку елементів Web-сторінки з використанням синтаксису CSS та XPath. Наприклад, щоб застосувати деяке правило CSS для усіх елементів, достатньо виконати команду: *\$('*').css('color', 'blue')*. Інші приклади:

`$("input:checked").length` – повертає кількість елементів форм, які виділені;

`$("td:empty").text("порожній").css('background', 'red')` – знаходить усі комірки таблиці `<td>` за умови, що вміст їх порожній (псевдоклас *empty*), після цього вносить в ці комірки текст «порожній» та встановлює для них колір фону в червоний.

`$("#d1,#s1,#par24").css('border','1px dashed black')` – знаходить елементи з ідентифікаторами *d1*, *s1*, *par24* та встановлює для них пунктирну границю розміром 1px чорним кольором;

`$('input[value="321"]').val("123")` – повертає усі елементи форм, в які уведено значення «321» і замінює це значення на «123»;

`$('div[id]').text("My text")` – знаходить усі елементи `<div>`, що містять атрибут *id* та заповнює їх вміст текстом «My text».

Методи уточнення та обробки результатів відбору елементів JQuery

Після відбору елементів засобами селекторів, бібліотека JQuery має можливість додатково відфільтрувати результат:

а) *children()* – отримати усіх нащадків для знайдених елементів:

`$("#d1").children().css('background-color', 'red')` знаходить усіх нащадків елемента з ідентифікатором *d1* та застосовує для них правило CSS;

б) *filter(x)* – відфільтрувати елементи, використовуючи замість «x» функцію, інший селектор або елемент JQuery:

```
$('p').filter(function(index) {  
    return $(this).text().length  
    >3;  
}).css('background-color', 'red')
```

В наведеному прикладі знаходяться усі параграфи, до них додатково застосовується функція фільтрації, яка перевіряє довжину тексту всередині. Якщо довжина більша трьох, то для таких параграфів буде застосовано правило CSS;

в) *each()* – виконати ітерацію по усіх знайдених елементах:

```
$('#div').each(function(index) {  
    alert(index + ': ' + $(this).text());  
});
```

В даному прикладі буде знайдено усі елементи *<div>* та для кожного з них застосується функція, яка виведе номер елемента та його вміст.

г) *map()* – застосувати функцію для кожного зі знайдених елементів: *\$('#:input').map(function() { return this.id; }).get().join(',')*;

В цьому прикладі буде знайдено усі елементи форм, в кожного з них буде взято ідентифікатор і з них буде сформовано рядок виду: *id1,id2,id3*; де *id1,id2,d3* – ідентифікатори елементів.

Маніпуляція вмістом Web-документа

До групи методів маніпуляції елементами сторінки відносяться: додавання/вилучення властивостей CSS, встановлення/очищення текстового вмісту елемента та його дочірніх елементів тощо.

1) *css(property)*, *css(property,value)* – отримує та встановлює значення властивості CSS відповідно:

```
var a = $('#e1').css(„position“) // взяти значення position для елемента з  
id=e1
```

```
$('#,p').css(„position“,a) // встановити усім параграфам значення  
властивості position, як у змінній a
```

2) *html()*, *html(value)* – отримує та встановлює значення HTML-вмісту, включаючи теги та вміст дочірніх елементів, відповідно:

```
alert($('#a1').html()) // вивести вміст елемента з  
id=a1
```

```
$('#,a1').html(„<span>123</span>“) // встановити вміст елемента з  
id=a1
```

значенням *123* (створити елемент ** всередині *a1*)

3) *remove()* - вилучити знайдені елементи:

```
$('.c1, #e1').remove(); // вилучити наступні елементи: у яких клас = c1; з  
id=e1
```

3) *empty()* - очистити знайдені елементи:

`$(p').empty();` // очистити вміст усіх параграфів

4) *`after()`, `before()`* –вставити вміст після/до знайдених елементів:

`$('#d1').after('000')` // вставити текст 000 після елемента з `id=d1`

`$('#d1').before("Link")` // вставити гіперпосилання перед елементом з `id=d1`

5) *`text()`, `text(value)`* – отримує та встановлює значення текстового вмісту, без тегів, але із текстовим вмістом дочірніх елементів, відповідно:

`alert($('#a1').text())` // вивести вміст елемента з `id=a1`

`$('#a1').text('123')` // встановити вміст елемента з `id=a1`

значенням `123` (елемент `` не створюється, вставляється лише текст «`123`»)

Оброблювачі подій

Завдяки оброблювачам подій програміст має змогу застосовувати той чи інший програмний код, реагуючи на дії користувача (події від миші, клавіатури, операцій введення/виведення). Розглянемо способи встановлення оброблювачів подій:

1) *`bind(eventType [, eventData], handler(eventObject))`* – універсальний метод встановлення оброблювача події до знайдених елементів, де *`eventType`* – тип події, яка оброблюється, *`eventData`* – необов'язковий параметр, який містить дані, що передадуться оброблювачу, *`handler(eventObject)`* – функція-оброблювач події:

`$(p').bind('click', function() { alert($(this).text()); });` // для усіх `<p>`

встановити як оброблювач події натискання на ліву кнопку миші функцію, яка виводить вміст елемента

`$(p').bind('click',function(e){alert(e.pageX+';'+e.pageY)})` - // для усіх `<p>`

встановити як оброблювач події натискання на ліву кнопку миші функцію, яка виводить координати курсору в точці, де відбулось натискання

Зауваження. „e” – параметр, який передається будь-якому оброблювачу і уточнює подію, містить властивості про натиснуту клавішу, координати курсора, про тип елемента та інші.

2) *click()*, *dblclick()*, *focus()*, *blur()*, *change()*, *load()*, *unload()* – методи для обробки найпоширеніших подій: натискання та подвійного натискання кнопки миші, встановлення/втрати фокусу елемента, зміни його вмісту, завантаження/вивантаження елемента (зображення, сторінки).

```
$(':input').change(function() {  
    alert('вміст змінено');  
});
```

3) *unbind([eventType] [, handler(eventObject)])* – відміння використання оброблювача події, де *eventType* – тип події, яка оброблюється, *handler(eventObject)* – функція-оброблювач події:

```
$('div').unbind('click'); // усім <div> відмінити обробку події onClick
```

```
var handler = function() {  
    alert('оброблювач');  
};
```

```
$('#a1').bind('click', handler);
```

```
$('#a1').unbind('click', handler); // handler – вказівник на функцію, що  
використовувались при bind
```

4) *\$(document).ready(handler)* – оброблювач події «завантажено DOM-модель».

Даний тип подій застосовується замість події `—onload` елемента `<body>`:
`<body onload="alert(1)"> ... </body>`

На відміну від `—onload` оброблювач `—ready` виконується раніше –
в

момент, коли сформовано дерево DOM об'єктів сторінки. В той же час
`—onload`

викликається пізніше, а саме після завантаження усіх сторонніх об'єктів
сторінки: зображень, флеш-компонентів тощо. Приклад оброблювача


```
$(document).ready(function() {  
    alert(„Сторінка завантажена”)  
});
```

Асинхронний обмін інформацією з Web-сервером. Ajax

Ajax (асинхронний Javascript) – це підхід до створення інтерактивних Web-додатків, згідно з яким обмін даними між Web-браузером та Web-сервером відбувається у фоновому режимі та надає можливості оновлювати лише частину Web-сторінки. Такий підхід зменшує мережний трафік та дозволяє будувати додатки, наближені до «віконних», які є більш звичними для користувача. Головним чином підхід базується на використанні засобів мови

Javascript. При цьому з серверного боку жодних змін виконувати не потрібно у порівнянні з традиційним підходом. На клієнті за Ajax відповідає об'єкт

Javascript XMLHttpRequest. Передача даних за допомогою цього об'єкта може відбуватись як у синхронному, так і у асинхронному режимі. Складність безпосереднього використання цього об'єкта зумовлена неповною сумісністю реалізації в різних браузерах. Тому, як правило, використовують крос-

платформні бібліотеки, які уніфікують доступ до даного об'єкта. Однією з таких бібліотек є JQuery. З огляду на задачі Ajax JQuery пропонує наступні можливості:

1. Завантаження даних з сервера безпосередньо у елемент сторінки (наприклад, у параграф) – метод *\$.load()*.
2. Завантаження та виконання програмного коду мовою Javascript – метод *\$.getScript()*.
3. Завантаження даних із сервера у форматі JSON, що спрощує перетворення даних у об'єкти Javascript – метод *\$.getJSON()*
4. Відправлення даних на сервер - метод *\$.post()*
5. Обробка помилок, що виникли при передачі даних.
6. Сериалізація даних HTML-форми для подальшої передачі на сервер. *\$(<selector>).serialize()*

Методичні вказівки

Використання серверних Web-технологій

Для виконання даної лабораторної роботи необхідно додатково встановити Web-сервер. Рекомендовано використовувати Web-сервер Apache, виходячи з його широкого розповсюдження та простоти інсталювання. Для ОС сімейства Windows Web-сервер Apache являє собою повноцінний дистрибутив, встановлення якого потребує мінімальної участі користувача. При встановленні Apache треба переконатись, що TCP-порт комп'ютера з номером 80 вільний. Це можна зробити шляхом виклику із термінального вікна (*cmd*) команди

netstat -a -n -p TCP

В результуючому списку не повинно бути запису з портом 80. Для перевірки коректності встановлення Web-сервера необхідно у адресному рядку браузера (Internet Explorer, Mozilla Firefox або іншому) увести команду *http://localhost*. При цьому у відповідь на запит Web-сервер поверне список файлів його кореневого каталогу або сторінку привітання (в залежності від версії Web-сервера). Обидва результати свідчать про успішне встановлення.

Особливістю роботи Web-сервера Apache є його функціонування у вигляді служби Windows, що призводить до автоматичного запуску Web-сервера при завантаженні ОС Windows. Файли сайту такі, як html, php, css, js та інші зберігаються в спеціальному каталозі Web-сервера. За замовчуванням він знаходиться у тому ж місці, де і решта файлів Web-сервера – в каталозі *htdocs*.

Вміст саме цього каталогу виводиться за запитом *http://localhost* (рис.3.1).



Рис.3.1. Приклад виведення після успішного встановлення Apache

За необхідності шлях до цього каталогу можна змінити, для цього треба відкоригувати файл налаштувань Web-сервера *httpd.conf*, який знаходиться в підкаталозі *conf* Web-сервера. За кореневий каталог відповідає параметр *DocumentRoot*. Після корекції даних конфігурації Web-сервер необхідно перезавантажити через панель управління Windows (наприклад, Адміністрування-Служби-Apache2.2). Файл конфігурації, крім цього, дозволяє налаштувати віртуальні хости (комп'ютери), обмежити доступ до файлів та каталогів сайту, налаштувати переадресацію, а також підключити модулі для обробки певного типу запитів.

Для виконання роботи достатньо перенести усі файли (*html, js, css, txt*) у каталог *htdocs*.

Застосування Ајах для асинхронного завантаження XML-документа

В задачах Web-програмування часто виникає необхідність обробки XML-документів. Прикладом може служити динамічне багаторівневе меню, ієрархічний каталог деяких об'єктів, зокрема, товарів за категоріями, фрагмент файлової системи тощо. Перевагою бібліотеки JQuery є наявність в ній вбудованих засобів роботи з XML в той же спосіб, що і зі звичайною Web-сторінкою: є можливість відбирати елементи XML-документа, використовуючи синтаксис CSS та XPath, застосовувати до них операції отримання/встановлення вмісту, атрибутів тощо.

Нехай на сервері розташовано XML-документ (data.xml) наступного вигляду:

```
<people>
  <man id="m1">
    <name
id="n1">John</name>
  </man>
  <man id="m2"/>
  <man id="m3"/>
</people>
```

Необхідно асинхронно завантажити документ, вивести ідентифікатори вузлів та вміст елементів `<name>`.

Для реалізації даної задачі будемо використовувати найбільш універсальну функцію JQuery з назвою `.Ajax()`. Вона дозволяє встановити метод HTTP за яким треба виконати запит на сервер, тип ресурсу, який запитується, описати функцію успішного завантаження та встановити інші параметри:

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script>

$(document).ready(function()
{ $.ajax({
  type: "GET",
  url: "data.xml",
  dataType: "xml",
  success: function(xml) {
    $(xml).find('man').each(function(){
      document.writeln($(this).attr("id"))
      $(this).find('name').each(function(){
document.writeln($(this).text())}
      ) })
    }
  }
});
```



```

        }
    })
})
</script>
</head>
<body>
</body>
</html>

```

В цьому прикладі описано оброблювач події `ready()`, який викликається автоматично при завантаженні сторінки. При цьому виконується запит функцією `ajax`. При успішному виконанні запиту (властивість `success`) в оброблювач буде передано параметр `xml`, який і буде відображенням файлу `data.xml`. Всередині цієї функції виконується прохід по елементам 1-го рівня `$(xml).find('man').each(...)` із виведенням значення атрибута `id` кожного з елементів. Крім того, для кожного елемента 1-го рівня виконується пошук елементів з атрибутом `name` і виведення значення його `id`. В результаті у вікні браузера буде виведено: `m1 John m2 m3`.

Застосування Ajax для асинхронного завантаження об'єкта Javascript

В Javascript для зручності обробки використовується спеціальний формат подання об'єктів цієї мови JSON (об'єктний запис Javascript). Його особливістю є зберігання об'єктів у вигляді пар: ключ/значення. Ключ – це рядок, який автоматично перетвориться у назву властивості об'єкта; значення – рядок, об'єкт, масив Javascript. Очевидною перевагою JSON-формату у порівнянні з XML є компактне подання даних. Для прикладу розглянемо задачу завантаження персональних даних про особу (ім'я та його вік) у поля форми XHTML. Дані зберігатимемо на сервері у файлі `user.json`. Вміст цього файлу буде наступним:

```

{
  "data": [

```

```

    {
        "name": "user1",
        "age": "20"
    },
    {
        "name": "user2",
        "age": "19"
    }
]
}

```

Фігурні дужки в файлі *user.json* означають запис об'єкта. Всередині – під ключем —*data* зберігаються дані про осіб, при чому значення подано у вигляді списку (масиву), в якому кожний елемент – це об'єкт з двома властивостями: *name* (ім'я), *age* (вік). Реалізує задачу наступний код HTML/Javascript:

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script>
function f(n){
$.getJSON('user.json', {}, function(json) {
    $("#username").val( json["data"][n].name
    ) $("#userage").val( json["data"][n].age )
})
}
</script>
</head>
<body>
<form>
    <input type="text" value="" id="username" />
    <br/> <input type="text" value="" id="userage" />
    <br/>

```



```

</form>

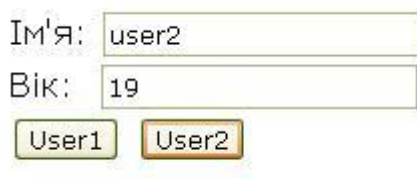
<button value = "User1"
onclick="f(0)">User1</button> <button value =
"User2" onclick="f(1)">User2</button>

</body>
</html>

```

В даному прикладі буде виведено два елемента форми та дві кнопки. При натисканні першої кнопки заповнюються дані про першу особу, при натисканні другої – про іншу.

Виведення буде наступним (рис.3.2).



Ім'я: user2

Вік: 19

User1 User2

Рис.3.2. Виведення програми-прикладу JSON

При натисканні на кнопку *User1* або *User2* викликається функція *f()*, де виконується запит до сервера, завантажуються дані з файлу *user.json* та формується об'єкт мови Javascript, який передається функції-оброблювачу. На основі вмісту цього об'єкту встановлюються відповідні значення полів *username* та *usage*. Оскільки в JSON-файлі використовувався масив, тому можемо звертатись до даних по індексу: *json["data"][n].name*.

Функції \$.get(), \$.post() та \$(selector).load()

Дані функції є скороченнями функції *ajax()* і призначені відповідно до завантаження (відправлення) даних відповідно методами GET або POST, а також методом GET безпосередньо у елемент.

Наприклад, задано три елемента *div*, два елементи уведення форми та кнопка відправлення даних на сервер (рис.3.3):



Рис.3.3. Початкова форма введення прикладу *\$.get*, *\$.post()*, *\$(sel).load()*

При натисканні на текст «*click d1*» буде використано метод *\$.get()*, при натисканні на *d3* – *load()*, а при натисканні на кнопку Send будуть відправлені на сервер дані з форми методом *\$.post()*, сервер їх поверне в тому ж вигляді, а оброблювач *\$.post()* розмістить результат всередину *d2* (рис.3.4).

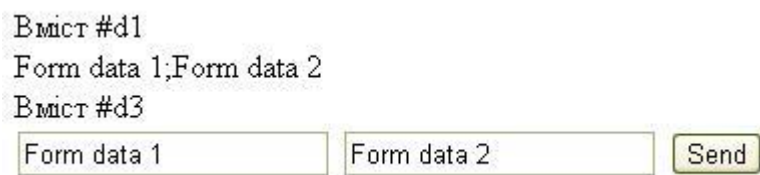


Рис.3.4. Результат виконання методів *\$.get*, *\$.post()*, *\$(sel).load()*

Вихідний текст *html* та *javascript* ілюструє наступний фрагмент:

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script>
function f()
{
$.post('form.php', $('#frm').serialize(), function(data) { $("#d2").html(data) })
return false;
}
</script>
</head>
<body>
<div id="d1" onclick="$.get('1.txt', function(data) { $('#d1').html(data)
})">click d1</div>
<div id="d2">Send the form data...</div>
<div id="d3" onclick="$('#d3').load('3.txt')">click d3</div>
```

```
<form id="frm" onsubmit="return f()">
  <input type="text" id="i1" name="i1" />
  <input type="text" id="i2" name="i2" />
  <input type="submit" value="Send" />
</form>
</body>
</html>
```

Особливістю даного фрагмента є передача даних з форми на сервер без перезавантаження основної сторінки. В прикладі для цього використано запис вигляду: `$.post('form.php', $('#frm').serialize(), function(data) { $('#d2').html(data) })`. Тут викликано файл *form.php* на сервері (його вміст: `<? echo $_POST["i1"].".".$_POST["i2"]; ?>`, що означає повернення отриманих даних клієнту). Як другий параметр передано результат функції *serialize()*, застосованої до форми *frm*. *serialize()* – функція, яка збирає усі значення елементів форми у єдиний рядок, розділений символом `&` – в тому ж форматі, як і при відправленні методом GET на сервер. Третій параметр методу *post* – функція обробки результату, отриманого з сервера. В даному випадку остання заповнює вміст елемента `<div id="d2">`.

Варіанти завдання

Завдання виконуються згідно з таблицею варіантів бригадами не більш, ніж по 2 студенти в кожній. Код JavaScript (jQuery), HTML та CSS має зберігатися у окремих файлах. Джерелом даних є XML-файли, які зберігаються на Web-сервері.

Особливості оформлення протоколу. Протокол має включати: титульний аркуш, текст завдання, структуру файлу XML у вигляді XML DTD, фрагмент програмного коду мовою Javascript, а також 2-3 копії екранних форм.

Оцінювання роботи. При оцінюванні роботи враховується: рівень відповідності вимогам до завдання, оформлення роботи, якість програмного коду та структури XML-файлу з даними.

Додаткове програмне забезпечення. Для отримання даних, які зберігаються у файлах XML на сервері, слід встановити Web-сервер Apache 2.2. Для зберігання даних, отриманих від клієнта з HTML-форм та збереження їх у файлі, використовувати мову PHP 5.

Варіант завдання визначається викладачем (див. табл. 3.1).

Таблиця 3.1

Варіанти завдань

№	Назва завдання	Вимоги щодо виконання роботи
1	2	3
1.	Інтерактивна HTML-форма	Реалізувати можливість інтерактивної перевірки коректності заповнення HTML-форми реєстрації користувача. Включити такі елементи: 1) довжина та складність пароля; 2) перевірка наявності імені користувача в системі; 3) інтерактивна підказка країни та міста мешкання. Поля та їх типи для перевірки та підказки про помилки взяти з XML-файлу. Усі перевірки виконувати (і сигналізувати в разі помилки) інтерактивно. Результати перевірки заносити в інший XML-файл.
2.	Аjax-таблиця	Реалізувати можливість редагування та перегляду таблиці HTML із полями: ціле, символний рядок, елемент зі списку. Включити можливість вставки, вилучення та редагування комірок таблиці. Синхронізувати кожну дію з інформацією у XML-файлі.
3.	Аjax-дерево	Реалізувати інтерактивне завантаження гілок дерева (наприклад, каталогу документів) з бази даних, при виборі визначеної гілки відобразити текст Web-сторінки. Структуру дерева, його вміст та відповідні гілкам HTML-сторінки зберігати у XML-файлі. Навігація по дереву і відображення HTML-сторінки відбувається без перезавантаження основної сторінки.
4.	Аjax-редактор текстових файлів	Надати можливість створення та редагування умовних текстових файлів з сервера. Передбачити виведення списку вже існуючих файлів. Файли зберігати на сервері всередині XML-документа. Реалізувати функції перегляду, запису та вилучення файлів. Операції пересилання даних виконувати асинхронно.

1	2	3
5.	Аjax-електронна пошта	Надати можливість обміну повідомленнями між двома користувачами. Передбачити такі функції, як перевірка поштової скриньки, створення та відправлення нового листа, перегляд та вилучення існуючого листа. Листи зберігати на сервері в XML-файлі. Операції пересилання даних виконувати асинхронно. Реалізувати функцію входу користувача за паролем.
6.	Аjax-щоденник	Реалізувати можливість ведення інтерактивного щоденника із можливістю створення нових тем, відповідних повідомлень та врахування дати їх створення. Крім того, реалізувати можливість перегляду повідомлень та їх вилучення. Дані зберігати у вигляді XML-файлі на сервері.
7.	Аjax-фотогалерея	Реалізувати можливість створення фотогалереї із функціями додавання та вилучення зображення, а також перегляду наявних фотографій як окремо, так і у вигляді слайд-шоу. Файли зображень завантажувати на сервер, інформацію про список файлів зберігати у XML-файлі на сервері.
8.	Гістограма	Реалізувати можливість побудови гістограми засобами CSS. Кожний стовпчик гістограми будувати на основі генерації псевдовипадкового числа (або при уведенні користувачем), який додається у файл на сервері у форматі XML у асинхронному режимі. За сигналом від таймера оновлювати зображення гістограми.
9.	Аjax-авторизація користувачів	Реалізувати можливість створення, вилучення та блокування облікових записів користувачів (ім'я, пароль) і віднесення їх до однієї з трьох груп. У відповідності до групи при авторизації користувача видавати йому одну з трьох HTML-сторінок. Усі операції виконувати без перезавантаження сторінки. Дані про користувачів зберігати у XML-файлі на сервері.
10.	Аjax-словник	Реалізувати можливість інтерактивного створення, наповнення та вилучення слів у словнику, який зберігається на сервері у вигляді XML-файлу. Окремий запис словника являє собою пару слів «слово українською мовою» - «слово англійською мовою». Забезпечити можливість перекладу слів в обидва напрями без перезавантаження сторінки.

1	2	3
11.	Аjax-персоналізатор сторінки	Реалізувати можливість персоналізації сторінки за запитом користувача. Передбачити можливість зміни фонового зображення сторінки, шрифту текстових фрагментів та заголовків сторінки. Крім того, дозволити користувачам створювати нові стилі, на основі передбачених властивостей. Дані зберігати в XML-файлі на сервері. Операції пересилання даних виконувати асинхронно.
12.	Аjax-редактор секцій сторінки	Реалізувати можливість переміщення основних блоків сторінки за запитом користувача. Основні блоки: меню, основний вміст, нижній колонтитул. Створити декілька схем виведення блоків і надати можливість користувачеві обирати одну из них. Крім того, дозволити користувачеві запам'ятовувати свій вибір для подальшого відновлення. Дані зберігати в XML-файлі на сервері. Операції пересилання даних виконувати асинхронно.
13.	Аjax-планувальник справ	Реалізувати можливість створення списку нагадувань щодо справ на день. Передбачити функцію візуального сповіщення про настання запланованої події. Кожна подія включає наступні поля: назва, текст, запланований час. Дані зберігати у вигляді XML-файлу на сервері, передачу даних виконувати асинхронно.
14.	Аjax-меню	Реалізувати динамічне формування і завантаження меню та відповідної Web-сторінки без перезавантаження основної сторінки. Структуру меню, включаючи вкладені рівні необхідно завантажувати з XML-файлу.
15.	Аjax-закладки	Сформувати 5 закладок на основній Web-сторінці, кожна з яких відповідає окремій сторінці. При переході на закладку необхідно завантажити її вміст без перезавантаження основної. Назви, кількість та вміст закладок брати з XML-файлу на сервері.