МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ПРАКТИЧНИХ ЗАНЯТЬ З ДИСЦИПЛІНИ «ОСНОВИ WEB-ПРОГРАМУВАННЯ»

для студентів ОС «бакалавр» денної та заочної форм навчання
галузі знань 12 Інформаційні технології
спеціальності 122 Комп’ютерні науки

Луцьк-2023

УДК 004.43(072)

М. 54

Методичні вказівки до виконання практичних занять з дисципліни «Основи Web-програмування» для студентів ОС «бакалавр» денної та заочної форм навчання галузі знань 12 Інформаційні технології спеціальності 122 Комп’ютерні науки [Електронний ресурс] / укл. В.І. Костін. – Луцьк : ДонНТУ, 2023. – 45 с.

Методичні вказівки до виконання практичних занять з дисципліни «Основи Web-програмування» містять методичні рекомендації щодо виконання завдання з практичних занять, рекомендовану літературу. Призначені для студентів ОС «Бакалавр» усіх форм навчання галузі знань 12 Інформаційні технології спеціальності 122 Комп’ютерні науки.

Укладач: В.І. Костін, ст. викл. кафедри прикладної математики та інформатики

Рецензент: Олександр ШТЕПА , к.т.н., доцент, доцент каф. ЕТ

Відповідальний за випуск: Наталія Маслова, к.т.н., доцент, в.о. зав. кафедри прикладної математики та інформатики

Затверджено навчально-методичним відділом ДВНЗ «ДонНТУ»,
протокол № 7 від 25.04.2023 р.

Розглянуто на засіданні кафедри прикладної математики та інформатики, протокол № 4 від 18.04.2023 р.

ЗМІСТ

ВСТУП

Метою вивчення навчальної дисципліни «Основи Web-програмування» є набуття студентами знань про Web-програмування та навичок програмування Web-сайтів і Web-інтерфейсів; вивчення засобів програмування серверних скриптів (CGI), освоєння можливостей мов JavaScript та CGI, а також створення локальних скриптів з використанням мови JavaScript.

Практичні заняття з дисципліни «Основи Web-програмування» є складовою частиною навчального процесу підготовки бакалаврів з інформаційних технологій. Виконання практичних занять служить для закріплення студентами знань, отриманих в процесі вивчення дисципліни, та грунтується на вмінні студентів працювати зі спеціальною літературою.

Практичне заняття №1. Організація представницької Web-сторінки на основі HTML

Завдання

1. За погодженням з викладачем вибрати тематику згідно таблиці 1. Підготовлені HTML-документи повинні містити цілісну інформацію та бути закінченим інформаційним проектом.

2. Підготувати щонайменше 3-х взаємозалежних WWW-документів, які мають обраний об'єкт, і файли графічних образов:

- іконки (5 шт. формату bmp або gif);
- середніх розмірів (2 шт. формату gif);
- великих розмірів (1 шт. формату jpg або gif).

3. Завдання з текстової частини:

У документах обов'язково повинні бути використані такі елементи мови HTML:

- різні типи заголовків з різним типом вирівнювання;
- виділення ділянок тексту кількома шрифтами (логічними та фізичними, не менше трьох кожного виду),
- параграфи, роздільники;
- посилання:
  – усередині документа;
  – на інші документи поточного каталогу;
  – на інші документи іншого каталогу;
  – на документи, що знаходяться на віддалених терміналах;
  – списки:
- упорядковані списки;
- неупорядковані списки;
- списки словники (глосарії).

Списки повинні включати підписки з різною мірою вкладеності і різних видів з різними типами позначення елементів списків. Деякі елементи списку мають бути гіперпосиланнями.

Таблиця 1 - Теми сайтів

Варіант	Бібліотека	Тема
1		Індійський стиль програмування
2		Огляд мов програмування для написання Web-скриптів
3		Silverlight
4	Canvas	Графічний тег HTML-5
5	Графічні системи мовою JavaScript	JSXGraph - бібліотека графічної візуалізації на Javascript
6	Графічні системи мовою JavaScript	DHTML, JavaScript Draw Line, Circle, Ellipse (Oval), Polyline, Polygon, Rectangle
7	Графічні системи мовою JavaScript	Raphael - JavaScript Library
8	Графічні системи мовою JavaScript	Плагін jqPlot - побудова діаграм та графіків
9	Графічні системи мовою JavaScript	WebFX - The Web Graphics
10	Бібліотеки мовою JavaScript	Prototype
11	Бібліотеки мовою JavaScript	Modernizr
12	Бібліотеки мовою JavaScript	Scriptaculous
13	Бібліотеки мовою JavaScript	Dojo
14	Бібліотеки мовою JavaScript	ExtJS
15	Бібліотеки мовою JavaScript	JQuery
16	Бібліотеки мовою JavaScript	Mootools
17	Технологія Ajax	Основи Ajax
18	Технологія Ajax	Методи використання Ajax
19	Технологія Ajax	Налагодження Ajax-додатків
20	Системи керування контентом	
21	Системи керування контентом	Joomla!
22	Системи керування контентом	Joomla! Розширення
23	Системи керування контентом	Joomla! Компоненти та модулі
24	Системи керування контентом	Dropa. Робота з базами даних
25	Системи керування контентом	Dropa. Призначення та основи роботи

4. Завдання з графічної частини:

- включити в документи вбудовані графічні образи (gif, jpg, png) з альтернативним текстом та параметрами ширини та висоти.

- включити в документи іконки для позначок елементів списків та завдання посилань зі зміною розмірів зображень, що виводяться.
- один із вбудованих образів використовувати як картку для посилань на інші документи. Під час підготовки бази даних карти виділити ділянки образу з допомогою прямокутників, кіл і багатокутників.

5. Розробку та початкове тестування документів виконати в середовищі Windows за допомогою браузерів або інших редакторів HTML.

6. Завдання за таблицями:

Включити до документів щонайменше 2-х таблиць із різним числом рядків і шпальт у них. У таблицях мають бути:

- заголовки таблиці за стандартом оформлення звітів;
- назви стовпців таблиці;
- осередки таблиці різних розмірів (у кілька рядків або стовпців);
- простий текст із "загортанням" і без "загортання";
- гіпертекстові посилання;
- списки;
- зображення активних образів.

7. Завдання з кадрів:

- включити до документів не менше 3-х фреймових документів та організувати незалежне прокручування їхнього вмісту.
- розміри фреймів повинні задаватися в наступному вигляді:
- відносний розмір кадру у відсотках;
- відносний розмір кадрів з урахуванням ваги;
- абсолютний розмір у пікселях;
- розділити один кадр на кілька інших орієнтації з різними можливостями прокручування кадру;
- при організації гіперпосилань має виконуватись:

– зміна вмісту вікна кадру, що включає саму гіпертекстову посилання;

– зміна вмісту іншого кадру;

– зміна вмісту батьківського кадру;

– зміна вмісту всього вікна браузера.

8. Завдання за формами:

- за погодженням з викладачем вибрати інформацію, що подається в HTML-формі та спосіб її обробки на сервері.
- розробити HTML-форму на додаток до раніше створених WWW-документів, що включає:

– кілька полів для введення тексту, паролю та для введення числа;

– кілька груп кнопок типу “checkbox”;

– кілька груп кнопок типу "radio";

– поле багаторядкового текстового введення;

– вибір <select> зі списку та розкривного меню з єдиним та множинним вибором;

– вибір файлу;

– файли графічних образів;
– кнопки скидання та передачі;
– об'єднання елементів форми в групи;
– об'єднання опцій select в групи.

9. Застосувати каскадуваних таблиць стилів;

- застосувати властивості, що визначають фон і текст, що задають обрамлення елементів та списків;
- показати групування властивостей спрощення визначення стилю.

10. Сторінка про себе

- рік народження;
- яку школу закінчили і де;
- коротко про сім'ю;
- ваші захоплення;
- декілька фотографій про себе, друзів,сем'ю.

Теги мови HTML, що використовуються

Загальна структура документа: HTML, HEAD, BODY.
Заголовки та роздільники: TITLE, Hn, P, HR, BR, PRE.
Фізичні шрифти: B, I, TT, U та ін.
Логічні шрифти: сімейства шрифтів Ms, що підключаються через каскадні таблиці стилів.
Списки: UL, OL, LI, DL, DT, DD.
Мітки: <A NAME="label">.
Посилання: <A HREF="URL">Reference</A>.
Зовнішні : <A HREF="URL">
Вбудовані образи: <IMG SRC="URL">
Образи-посилання: <A HREF="URL"><IMG SRC="URL-IMG"></A>
Активні карти зображень: <A HREF="URL-CGI-MAP">
CELLPADDING=WEIGTH= BGCOLOR= > .....</TABLE>,
<IMG SRC="URL-IMG" ISMAP></A>
Таблиця: <TABLE BORDER= CELLSPASING=
Осередки таблиці: - <TD ALIGN= VALIGN= NOWRAP
WEIGTH=BGCOLOR= ROWSPAN= COLSPAN=> .... </TD>.
Рядок таблиці: - <TR ALIGN= VALIGN=>
Заголовок стовпця: - <TH ALIGN= VALIGN= NOWRAP
WEIGTH=BGCOLOR= ROWSPAN= COLSPAN= > ..... </TH>
Назва таблиці: - <CAPTION ALIGN=> Заголовок таблиці</CAPTION>

Фреймова структура: <FRAMESET ROWS= COLS= > ... </FRAMESET>
<NOFRAMES> ..... </NOFRAMES>
Параметри кадру: <FRAME
SRC=NAME=MARGINHEIGHT=MARGINWIDTH=SCROLLING=>
Зміна інших кадрів:
<A HREF="URL" TARGET="ім'я_мети">Гіпертекстове посилання</A>

Форма: <FORM ACTION="URL-CGI" METHOD="метод (GET I POST)" ENCTYPE="MIME type" >...</FORM>
Діалогові елементи форми:
- Введення рядка:
<INPUT TYPE="text" NAME="name" VALUE="value" SIZE=n MAXLENGHT=k>
- Введення числа:
<INPUT TYPE="number" NAME="name" VALUE="val" MIN=N_min MAX=N_max ERROR >
- Введення зображення:
<INPUT TYPE="image" NAME="name" SRC="URL_image">
- Вибір файлу
<INPUT TYPE="file" NAME="name" SIZE=n MAXLENGHT=k>
- Позначка параметрів:
<INPUT TYPE="checkbox" NAME="name" VALUE="val" CHECKED>
<INPUT TYPE="radio" NAME="name" VALUE="val" CHECKED>
<INPUT TYPE="button" NAME="name" VALUE="val">
- Введення рядків:
<TEXTAREA NAME="name" ROWS=N_rows COLS=N_cols WRAP>.....Початковий текст.....
</TEXTAREA>
- Вибір зі списку:
<SELECT NAME="name" SIZE=k NAME="name" MULTIPLE >
<OPTION VALUE="One" SELECTED> One
<OPTION VALUE="Two" DISABLED> Two
...
</SELECT>
- скидання форми:
<INPUT TYPE="reset">
- Подання форми на сервер:
<INPUT TYPE="submit">

Теги, які будуть враховуватися у цій роботі

1. Шрифти:
    1.1. Фізичні
    1.2. Логічні
2. Горизонтальна риса
3. Списки:
    3.1. Нумеровані арабськими цифрами
    3.2. Нумеровані римськими цифрами
    3.3. нумеровані буквами
    3.4. Ненумеровані
    3.5. Багаторівневі

4. Словник
5. Посилання:
   5.1. Всередині сторінки
   5.2. На іншу сторінку цього ж каталогу
   5.3. На сторінку в іншому каталозі
   5.4. На сторінку на іншому сайті
6. Зображення:
   6.1. Маленькі
   6.2. Середні
   6.3. Великі
   6.4. Міняючи розміри
   6.5. Вказівка альтернативної підпису
   6.6. Вказівка висоти і ширини
7. Активна карта:
   7.1. Коло
   7.2. Прямокутник
   7.3. Багатокутник
8. Таблиця:
   8.1. Мінімум 2 таблиці
   8.2. Заголовки таблиць
   8.3. Заголовки стовпців
   8.4. В осередках:
   8.4.1. Текст
   8.4.2. Посилання
   8.4.3. Зображення
   8.4.4. Підтаблиці
  8.5. Об'єднання декількох осередків в стовпці
  8.6. Об'єднання декількох осередків в рядку
9. Фрейми:
9.1. Мінімум 3 фрейма
9.2. Посилання:
9.2.1. Усередині фрейма
9.2.2. Інший фрейм
9.2.3. Батьківський фрейм
9.2.4. Головне вікно
9.2.5. Нове вікно
10. Форми:
   10.1. Текстове поле введення
   10.2. Багаторядкове текстове поле введення
   10.3. Введення пароля
   10.4. Множинний вибір
   10.5. Одиночний вибір
   10.6. Випадаючи список
   10.7. Складний список з одиночним вибором
   10.8. Складний список з множинним вибором

10.9. Вибір файлу
10.10. Кнопка скидання
10.11. Кнопка відправки форми
10.12. Об'єднання елементів форми в групи
10.13. Об'єднання опцій select в групи
11. Каскадні таблиці стилів:
11.1. Об'єднання бордюрів в таблицях
11.2. Колір шрифту
11.3. Розмір шрифту
11.4. Фоновий малюнок
11.5. Відстані в осередках таблиці:
11.5.1. Зверху
11.5.2. Знизу
11.5.3. Справа
11.5.4. Зліва
11.5.5. З усіх сторін
11.6. Власний маркер в списках
12. Сторінка про себе
12.1 Рік народження;
12.2 Яку школу закінчили і де;
12.3 Коротко про сім'ю;
12.4 Ваші захоплення;
12.5 Декілька фотографій.

Література: [1], [3], [5], [7], [8]

Практичне заняття №2
"Розробка CGI-сценарію для обробки форми HTML"

1. Розробити єдиний CGI-сценарій для обробки запитів, що надходять при заповненні форм за методами GET і POST, що повертає користувачеві відповіді у кодуванні **Windows-1251** у вигляді:

• текстовий документ;
• документа у форматі HTML;
• переадресація посилання з відповіддю користувачеві.

2. Форма береться з першої лабораторної роботи, що містить усі елементи, викладені у пункті 1b.

3. Для програмування скриптів застосовувати **не скриптові мови**: C, C++, Visual C, Delphi, TPascal, Python, Java та інших.

4. CGI-програма повинна зробити обробку даних форми. При обробці **обов'язково видавати назву елементів** форми та їх **вміст**.

5. Перевірку роботи скрипта проводити із застосуванням кирилиці.

Вказівки що до виконання заняття № 2

CGI (Common Gateway Interface) - загальний шлюзовий інтерфейс.

CGI-скрипт (сценарій) - програма, що написана відповідно до специфікації CGI будь-якою мовою програмування (C, C++, Pascal) або командною мовою (shell, cshell, командна мова MS DOS, Perl тощо).

Шлюз - це CGI-скрипт, який використовується для обміну даними іншими інформаційними ресурсами Internet або додатками-демонами. Звичайно CGI-програма запускається сервером HTTP для виконання деякої роботи, повертає результати серверу і завершує своє виконання. Шлюз виконується так само, тільки фактично він ініціює взаємодію клієнта і третьої програми.

Механізми обміну даних

Власне специфікація CGI описує чотири набори механізмів обміну даними:

- через змінні оточення;
- через командний рядок;
- через стандартне введення;
- через стандартний виведення.

Змінні оточення

Коли запускається зовнішня програма, сервер створює специфічні змінні оточення, через які передає програмі, як службову інформацію, так і дані. Всі змінні можна розділити на загальні змінні оточення, які генеруються при будь-якій формі запиту, і змінні, що орієнтовані на запит.

Слід відзначити, що оскільки на різних серверах імена змінних середовища, що використовуються для привласнення значень, можуть розрізнятися, то необхідно обов'язково перевірити ці імена в документації на певний сервер.

До загальних змінних відносяться:

- SERVER_SOFTWARE - визначає ім'я і версію програмного забезпечення серверу, яке відповідає на запит клієнта;
- SERVER_NAME - визначає доменне ім'я або IP-адрес сервера;
- SERVER_PROTOCOL - ім'я і версія інформаційного протоколу розроблялися не тільки для застосування в WWW з протоколом HTTP, але і для інших протоколів також, але широке застосування отримала тільки в WWW;
- SERVER_PORT - визначає номер порту TCP хост-машини, на якій працює сервер, за цим портом, за яким здійснюється взаємодія. За умовчанням для роботи на http використовується порт 80, але він може бути і переназначений при конфігуруванні серверу;
- REQUEST_METHOD - визначає метод доступу до інформаційного ресурсу. Це найважливіша змінна в CGI. Різні методи доступу використовують різні механізми передачі даних. Ця змінна може приймати значення: GET, POST, HEAD тощо;

- PATH_INFO - додаткова інформація про шлях, що передається в програму CGI. Передає розширення шляху до інформаційного ресурсу. Наприклад, в запиті;
  - http://144.206.192.100/cgi-bin/test?linear
  - розширення шляху - це "linear" і
  - PATH_INFO = linear.
- PATH_TRANSLATED - повний шлях до ресурсу. Річ у тому, що при конфігуруванні серверу деяким елементам (гілкам) дерева файлової системи можна призначити синоніми. Наприклад,

cgi-bin => /usr/local/etc/httpd/cgi-bin.

Тут праворуч - стандартне місце CGI-скрипта для серверу NCSA, а ліворуч - його синонім. При отриманні скриптом (сценарієм) test управління клієнт передає значення "cgi-bin/test", а в PATH_TRANSLATED буде

"/usr/local/etc/httpd/cgi-bin/test".

- SCRIPT_NAME - визначає ім'я і адресу сценарію, що виконується, так, як вказано клієнтом. Якщо параметри не вказані, то значення цієї змінної співпадатиме з тим, що клієнт передає як шлях, але якщо змінні вказані, то все, що слідує за знаком "?", буде відкинуто:

SCRIPT_NAME => "cgi-bin/search"

- QUERY_STRING - інформація про запит, що передана в програму. Для приєднання цієї інформації до URL використовується знак "?". Тобто ця змінна визначає зміст запиту до сценарію. Надзвичайно важливо при використовуванні методу доступу GET. В QUERY_STRING вміщається все, що записано після символу "?":

QUERY_STRING => "method+JavaScript"

При цьому ніякого перетворення рядка запиту сервером не проводиться. Все маніпулювання зі змістом QUERY_STRING можливо в сценарії.

Наступні дві змінні визначають тип і довжину інформації, що передається, від клієнта до серверу.

- CONTENT_TYPE - визначає MIME-тип даних, що передаються, сценарію. Використовуючи цю змінну, можна одним скриптом обробляти різні формати даних.
- CONTENT_LENGTH - визначає розмір даних в байтах, що передаються серверу, дана змінна надзвичайно важлива при обміні за методом POST, оскільки немає іншого способу визначити розмір даних, які треба прочитати із стандартного введення. Можлива передача і інших змінних оточення. В цьому випадку перед ім'ям вказується префікс "HTTP_".
- HTTP_ACCEPT - перелік медіа-типів, які може приймати клієнт.
- HTTP_FROM - адреса електронної пошти користувача, що направив запит (у багатьох браузерах ця змінна не підтримується).
- HTTP_REFERER - URL документа, на який клієнт вказує перед зверненням до CGI-програми.

- HTTP_USER_AGENT - браузер, яким клієнт користується для видачі запиту.

Окремий випадок представляють змінні, що породжуються в заголовку HTML-документа в дескрипторах META. Вони передаються в заголовку повідомлення, і деякі сервери можуть породжувати змінні оточення з полів цих заголовків.

Опції командного рядка

В даний час практично не використовується

Формат стандартного введення

Стандартне введення використовується при передачі даних в скрипт методом POST. Обсяг даних, що передаються, задається змінній оточення CONTENT_LENGTH, а тип даних - змінній CONTENT_TYPE.

Якщо, наприклад, з HTML-форми треба передати запит типу A=B&B=C, то змінна CONTENT_LENGTH буде дорівнювати 7, а змінна CONTENT_TYPE=application/x-www-form-urlencoded, а першим символом в стандартному введенні буде символ "A". Треба пам'ятати, що кінець файлу не передається сервером і тому завершити читання треба відповідно числа прочитаних символів.

Формат стандартного висновку

Стандартний висновок використовується скриптом для повернення даних серверу. При цьому висновок складається із заголовка і власне даних. Підсумок роботи скрипта може передаватися клієнту без яких-небудь перетворень з боку серверу, якщо скрипт забезпечує побудову повного HTTP-заголовка, інакше сервер заголовок модифікує відповідно специфікації HTTP. Заголовок повідомлення повинен відокремлюватися від тіла повідомлення порожнім рядком.

Зазвичай в скриптах вказується наступні поля HTTP-заголовка:

- Content-type;
- Location.

Поле "Content-type" вказує, коли скрипт генерує документ одразу і повертає його клієнту. В цьому випадку реального документа у файловій системі серверу не залишається. При використовуванні таких скриптів треба враховувати, що не всі сервери і клієнти відпрацьовують так, як представляється розробником скриптів. Так при вказівці Content-type:text/html, деякі клієнти не реалізують сканування отриманого тексту на предмет наявності в ньому вбудованої графіки. Зазвичай в Content-type вказують текстові типи text/plain і text/html.

Поле "Location" використовується для переадресації. Інколи переадресація допомагає подолати обмеження серверу або клієнта на обробку вбудованої графіки або попередньою серверною обробкою. В цьому випадку скрипт створює файл на диску і вказує його адресу в Location. Сервер, таким чином, передає реально файл, що дійсно існує.

Створення форми

<form атрибути> тіло форми </ form>

Атрибути:
- адреса програми обробки;
- метод передачі даних;
- тип даних.

Атрибут action = "URL" є обов'язковим. Найчастіше тут вказується логічний каталог, де знаходяться виконувані програми (скрипти). Атрибут method = "метод" є необов'язковим.

Найчастіше використовуються три методи передачі даних:
- GET,
- POST,
- HEAD.

За замовчуванням дані від форми йдуть по методу GET.

Атрибут enctype = "MIME-тип" також є необов'язковим.

Зазвичай використовується наступний тип з підтипом:

application/x-www-form-urlencoded.

Тут application - тип, а

x-www-form-urlencoded - підтип.

Види відповідей

1. **Скріпт видає тільки вміст.**

а) Content-type: text/html або Content-type: text/plain

б) Порожній рядок

в) <html> <head>

<title>

Опрацьований результат

</ title>

<body>

Вміст відповіді

</body>

</html>

2. Повна відповідь

Відповідь сервера на запит клієнта складається з трьох частин:
- рядок стану (Status-Line),
- заголовки протоколу HTTP,
  порожній рядок,
- тіло ресурсу.

Рядок стану

Рядок стану містить в собі:
- версію протоколу;

- код повернення;
- опис коду повернення.

Наприклад, HTTP / 1.0 200 Success

Класи кодів

Коди повернення діляться на п'ять класів:

- 1xx - інформаційні (що запит прийнятий і обробляється);
- 2xx - успішної передачі (запит успішно оброблений);
- 3xx - переадресації (запит не виконано. Його треба переадресувати);
- 4xx - помилки клієнта. Запит клієнта не повний. Від клієнта необхідна додаткова інформація:

це можуть бути:

- синтаксичні помилки при написанні запиту;
- у клієнта немає відповідних прав доступу до ресурсу і він повинен їх надати;
- документ не існує у даній адресою;
- не підтримується даний метод по даному URL;
- та ін.
- 5xx - помилки сервера (сервер зіткнувся з помилкою і, ймовірно, не зможе виконати запит клієнта. Дуже часто це помилки програміста, який написав скрипт).

Коди в діапазоні 1XX, 2XX і 3xx більшість Web-браузерів обробляють без сповіщення користувача. Код повернення позначає стан запиту, тобто чи був запит успішним чи ні, а також дані про причини неуспішного завершення запиту. Зазвичай цей код генерується сервером, але може і скриптом.

3. Переадресація

В цьому випадку йде наступний заголовок відповіді:

Location: /cgi/usr/my_path/my_doc.html

Порожня стрічка

CGI-програма повертає результати виконання сервера у вигляді потоку даних, що представляють собою (прямо чи опосередковано) відповідь на запит.

Потік даних складається з двох головних частин:

- заголовка;
- порожня стрічка;
- тіла повідомлення.

Заголовок складається з однієї або декількох рядків тексту і відділяється від тіла порожнім рядком.

Сервер шукає в потоці результатів наступні рядки заголовка:

Content-Type: значення

URI: <значення>

або

Location: значення.

Методи доступу

Метод - це HTTP-команда, з якою починається перший рядок запиту клієнта. Для HTTP визначені три основні методи: GET, HEAD і POST. Визначено і інші методи, але вони не так широко підтримуються серверами.

УВАГА! При завданні імен методів враховується регістр, тому GET і get будуть різними.

1. **Метод GET**

Дані від форми за цим методом додаються до URL скрипта, який обробляє ці дані. Тому дані, передані цим методом, зазвичай невеликі, так як обсяг переданих даних обмежений розмірами, відведеними під змінні оточення.

Існує різновид методу GET - умовний GET. При використанні цього методу сервер відповість на запит, тільки якщо будуть виконані умови передачі. Це дозволить розвантажити мережу, позбавивши її від передачі непотрібної інформації. Умова вказується в полі “if_Modified_Since" заголовка ресурсу.

2. **Метод POST**

POST - це метод розроблений для передачі великого обсягу інформації на сервер, а також для форм, які будуть що-небудь міняти або виконувати будь-які незворотні дії на сервері.

Ним користуються для анкетування існуючих ресурсів, посилки поштових повідомлень, роботи з формами інтерфейсів до зовнішніх баз даних і зовнішнім виконуваним програмам.

Дані в методі POST йдуть в такому ж форматі, як і для методу GET.

Формат цей наступний:

ім'я_поля1 = значення_поля1 &
ім'я_поля2 = значення_поля2 &
ім'я_поля3 = значення_поля3 & і т.д.

Всі символи тут, крім латинських букв, цифр і знака підкреслення, кодуються.

Символ "пробіл" замінюється знаком "плюс".

А всі решта символи замінюються знаком відсотка і двома шістнадцятеричними цифрами за ASCII таблиці.

3. Метод HEAD

HEAD - те саме, що і GET, але не повертається тіло ресурсу.

Використовується для отримання інформації про ресурси. Даний метод використовується для тестування гіпертекстових посилань, тобто діють вони чи ні. Умовного HEAD не існує.

Налаштування Apache-сервера

Інсталяція сервера – стандартна

Файли конфігурації (в каталозі conf)

Файл httpd.conf

а) шукається змінна ScriptAlias:

#ScriptAlias /cgi-bin/ "d:/Apache/cgi-bin"

де d:/Apache/cgi-bin - фізичний каталог, визначається при інсталяції.

Необхідно:

- перевірити відповідність реального каталогу з тим, що написаний в цьому рядку;
- прибрати коментар з цього рядка.

б) шукається змінна AddHandler:

#AddHandler cgi-script .cgi

де .cgi - вказує, які розширення вважаються скриптами;

- додати .exe, можна писати: .cgi .exe .pl або .cgi, .exe .pl
- розкоментувати цей рядок.

в) шукається змінна ServerName:

#ServerName new.host.name

Необхідно:

- розкоментувати рядок;
- встановити DNS-ім'я вашої машини в мережі або localhost.

Наприклад, ServerName http://localhost

Найпростіші скрипти

Приклад 1.

Даний скрипт написаний на командній мові Shell. Найпростіший має вигляд:

```
#!/bin/sh
echo Conteny-type: text/plain
echo
echo Тут результат виконання скрипта
# End script
```

У даному прикладі перший рядок визначає, що як інтерпретатор скрипт буде використовувати shell. Другий рядок відкриває заголовок повідомлення, передаваний скриптом серверу, і визначає тип передаваної інформації, як звичайний текст. Третій рядок відділяє тіло повідомлення від його заголовка. В тілі повідомлення передається фраза з четвертого рядка. Саме вона відображатиметься програмою-інтерфейсом користувача (браузером).

Приклад 2.

Як другий приклад приведений скрипт, що відображає значення змінних оточення:

```
#!/bin/sh
echo Conteny-type: text/plain
echo
echo $REQUEST_METHOD
```

```
echo $QUERY_STRING
echo $CONTENT_TYPE
echo $CONTENT_LENGTH
# End script
```

Тут користувачу будуть повернені значення вказаних в рядках echo змінних оточення.

Приклад 3.

Користувачу можна повернути не тільки значення змінних оточення, але і результати виконання команд:

```
#!/bin/sh
echo Conteny-type: text/plain
echo
finger oleg@pmi.donetsk.ua
# End script
```

У результаті виконання цього скрипта користувач отримає інформацію про користувача oleg з машини pmi.donetsk.ua.

Приклад 4.

Приклад скрипта, що написаний мовою С.

```
#include <stdio.h>                  /* Стандартний приклад. Програма, */
#include <stdlib.h>          /* що здійснює розбір рядка параметрів   */
                             /* отриманого з браузера HTML            */
#define MaxInLen 400   /* Максимальний розмір рядка параметрів*/
char Str[MaxInLen];      /* Рядок, що отриманий від браузера          */
char Str1[MaxInLen];    /* Тимчасовий буфер для розбору рядка    */
char Ch;
char *Tmp;
int I;
int M;
int ContLen;                  /* Довжина параметрів                    */
int TmpNum;
void main (void)
{
printf("Content-type: text/HTML\n\n");
Tmp = getenv("CONTENT_LENGTH");
/* Отримуємо довжину параметрів в текстовому вигляді                */
sscanf(Tmp,"%d",&ContLen);          /* Конвертуємо його в число     */
printf("<B>CONTENT_LENGTH</B> = %d<BR>\n", ContLen);
                                    /* Друкуємо довжину              */
printf("<B>Content from standart input:</B><BR>\n");

if (ContLen > MaxInLen) /* Якщо довжина параметрів більша, ніж
максимум, зайві символи буде втрачено                                */
  {
  ContLen = MaxInLen;
```

```
  printf("Input was cut to %d characters...<BR>\n", ContLen);
  }

ContLen++ ;
fgets(Str,ContLen,stdin); /* Отримуємо рядок з вхідного потоку      */
printf("<PRE>%s</PRE>", Str);              /* Друкуємо його          */
printf("<BR>\n<B>End Content ...</B><BR>\n");

printf("<BR>\n<B>Parsing ...</B><BR>\n");
                                    /* Починаємо розбиття рядка на окремі*/
for (I = 0, M = 0; I < ContLen;) /* параметри (поки залишаються       */
                                   /* символи в вхідному рядку)         */
  {
  if (Str[I]== '&')                /* Зустрівся роздільник параметрів   */
    {
    Str1[M]=0;              * Закінчити додання символів в параметр   */
    printf("%s<BR>\n", Str1);          /* Надрукувати параметр          */
    M = 0;
    I++;                             /* Перейти до наступного символу    */
    }
  else
    if (Str[I]== '%')                      /* % - ознака 16-ричного коду  */
      {
      Ch = Str[I+1];                 /* Формуємо число з текстового  коду*/
      if (Ch >= 'A') Ch -= 'A' - 10;
      else Ch -= '0';
      TmpNum = Ch << 4;
      Ch = Str[I+2];
      if (Ch >= 'A') Ch -= 'A' - 10;
      else Ch -= '0';
      TmpNum += Ch;
      Str1[M++]= TmpNum;       /* Записуємо в параметр символ       */
                                   /* з відповідним кодом               */
      I += 3;                        /* Перейти до наступного символу   */
      }
    else Str1[M++]= Str[I++];   /* Якщо просто символ – скопіювати  */
                                   /*  його без змін                    */
  }
Str1[M]= 0;                 /* Закінчити розбір рядка     параметрів      */
printf("%s<BR>\n", Str1);
printf("<BR>\n<B>End parsing ...</B><BR>\n");
}
```

В наведеному прикладі програма отримує дані через стандартне введення. Це означає, що використовується метод доступу POST. Число байтів, що одержуються, беруть із змінної оточення CONTENT_LENGTH.

Необхідно звернути увагу на те, що першим рядком запису в стандартний висновок є рядок, що визначає тип інформації. В даному випадку він визначений як текст HTML. Важливо також відзначити, що в цьому операторі висновок завершується двома переведеннями на новий рядок, оскільки порожній рядок повинен розділяти заголовок і тіло повідомлення, що передається серверу. Решта операторів виводить дані у вигляді HTML-пропозиції, формуючи HTML-документ.

При написанні сценаріїв слід враховувати те, що сервер звичайно стартує в мить, коли не всі шляхи можуть бути визначені, тому при зверненні до ресурсів слід вказувати повні шляхи для цих ресурсів.

Інтерфейс CGI можна застосовувати не тільки до обробки форм, але і до простих HTML-документів. Для цього на основній HTML-сторінці можна помістити CGI-посилання. URL буде переданий вказаній серверній програмі, яка одразу ж згенерує HTML-сторінку, що була викликана. Такий підхід можна використовувати для наведення даних, змінних в реальному часі, або для сторінок, які повністю побудовані на запитах до бази даних. Крім того, CGI можна використовувати для обробки карт зображень.

Література: [7], [8]

3. Загальні вимоги до практичних занять із застосуванням мови JavaScript

Практичні заняття із застосуванням мови JavaScript повинні містити листінг коду, в якому необхідно відображати наступне:

- титульний лист із назвою дисципліни, номер практичного заняття, його назву, варіант та конкретна умова роботи;
- лістинг програм та HTML-сторінок даної роботи;
- скріншоти вихідного та кінцевого (проміжного) станів;
- час, витрачений на виконання заняття.

У лістингах програм та на екранах виведення результатів занять мають відображатися:

- П.І.Б. виконавця та група;
- номер та назва заняття;
- варіант та умови заняття.

При розробці програм широко використовуватиме теги мови HTML та властивості каскадних таблиць стилів.

Практичне заняття №3
Розробка JavaScript-програми формування рядка, що біжить

1. Видати рядок, що біжить, у рядку титулу з документа, що розробляється (зліва направо).

2. Видати рядок, що біжить, у рядку титулу з документа, що розробляється (праворуч ліворуч).
3. Видати рядок, що біжить, у рядку титулу з двох кадрів (ліворуч праворуч).
4. Видати рядок, що біжить, у рядку титулу з двох кадрів (праворуч ліворуч).
5. Видати рядок, що біжить, у рядку титулу після заповнення текстового поля у формі (зліва направо).
6. Видати два рядки, що біжать, у горизонтальному вікні з документа, що розробляється (зліва направо).
7. Видати два рядки, що біжать, у горизонтальному вікні з документа, що розробляється (справа ліворуч).
8. Видати два різні рядки, що біжать, зліва направо і праворуч наліво в горизонтальному вікні з документа, що розробляється.
9. Видати два різні рядки, що біжать, зліва направо і праворуч наліво в горизонтальному вікні з іншого документа.
10. Видати два однакові рядки, що біжать, зліва направо і праворуч наліво в горизонтальному вікні з документа, що розробляється.
11. Видати рядок, що біжить, у рядку титулу після заповнення текстового поля у формі (праворуч ліворуч).
12. Видати рядок, що біжить, у горизонтальному вікні з документа, що розробляється (зліва направо).
13. Видати рядок, що біжить, у горизонтальному вікні з документа, що розробляється (праворуч ліворуч).
14. Видати рядок, що біжить, у горизонтальному вікні з іншого документа (зліва направо).
15. Видати рядок, що біжить, у горизонтальному вікні з іншого документа (справа ліворуч).
16. Видати рядок, що біжить, у горизонтальному вікні після заповнення текстового поля у формі (зліва направо).
17. Видати рядок, що біжить, у горизонтальному вікні після заповнення текстового поля у формі (праворуч ліворуч).
18. Видати рядок, що біжить, у рядку титулу з поточною датою і часом (зліва направо).
19. Видати рядок, що біжить, у рядку титулу з поточною датою і часом (справа ліворуч).
20. Видати рядок, що біжить, у горизонтальному вікні з поточною датою і часом (зліва направо).

Вказівки що до виконання заняття № 3

Для створення на сторінці сайту рядка, що біжить, можна скористатися різними способами:

- HTML (тег marquee);
- CSS;

- JavaScript;

У силу того, що завдання треба виконати способом JavaScript, то розглянемо як це можно зробити. Наприклад розглянемо рядок, що біжить справо наліва. Тут можна застосувати наступний алгоритм.

Вирізаємо перший символ поточного стану рядка і додаємо його в кінец рядка.

Приклад:

```
<h1>Видати рядок, що біжить, у полі (справо наліво). </h1>
<script>
  var dateStr = "Видати рядок, що біжить     ";
  var tStr;
  function Run()
{
  tString = dateStr.slice(1, dateStr.length) + dateStr.slice(0,1);
  dateStr = tStr;
  document.MyFrm.n1.value = tStr;
}
setInterval(Run,200);
</script>

<form name=MyFrm>
<input type="text" size=45 name=n1>
</form>
<script>
```

Якщо потрібно, щоб рядок бежав зліва направо, то в цьому випадку треба вирізати останній символ рядка і додати його до початку рядка.

Література: [2], [4], [5], [6]

Практичне заняття № 4. Розробка JavaScript-програми для показу властивостей різних об'єктів

1. Вивести список усіх гіперпосилань у документі різними шрифтами, що залежать від номера елемента у списку.
2. Вивести перелік всіх властивостей зазначеного об'єкта зображення.
3. Вивести спадаючий список усіх міток якорів документа та здійснити перехід на зазначену мітку після натискання кнопки.
4. Вивести список усіх властивостей зазначеного при введенні об'єкта у вигляді горизонтальної або вертикальної таблиць у документі різними шрифтами, що залежать від номера елемента у списку.
5. Вивести список усіх форм у документі у вигляді горизонтальної або вертикальної таблиць різними шрифтами, що залежать від номера елемента у списку.
6. Вивести список всіх елементів зазначеної під час введення форми.
7. Вивести перелік всіх елементів поточної форми.

8. Вивести список всіх елементів зазначеного при введенні спадного меню форми.
9. Вивести список усіх фреймів поточного документа.
10. Вивести спадаючий список назв усіх графічних зображень поточного документа.
11. Створити функцію, яка за 2-3 параметрами створює рядки таблиці, у яких у першому стовпці перебуває гіперпосилання, тоді як у другому - пояснення до них.
12. Вивести список вмісту властивостей обраного гіперпосилання у документі.
13. Вивести списки вмісту властивостей усіх гіперпосилань документа.
14. Вивести список усіх запроваджених аргументів для побудованої функції.
15. Вивести список усіх міток якорів яу документі різними шрифтами, що залежать від номера елемента у списку.
16. Видати список усіх властивостей зазначеного під час введення об'єкта певного кадру.
17. Видати список усіх властивостей зазначеного при введенні об'єкта зазначеного кадру.
18. Видати інформацію (номер і текст) про вибрані елементи у документі з кількох груп меню.
19. Розробити фреймову структуру, в якій текст будь-якого файлу одного фрейма міг розгортатися на значну частину екрана по висоті та назад за допомогою кнопок.
20. Вивести список усіх гіперпосилань у документі різними шрифтами, що залежать від номера елемента у списку та здійснити перехід на зазначену мітку після натискання кнопки.
21. Вивести спадаючий список усіх властивістей зазначеного зображенння поточного документа.

Вказівки що до виконання заняття № 4

Для отримання властивостей об'єкта застосовується оператор цикла for .... in.

Цей різновид циклу for використовується для реалізації циклу за властивостями об'єкта, який задається змінною objectName.

Синтаксис:

```
for (ind in objectName)
{
операторu
}
```

Індексна змінна *ind* пробігає по всім назвам властивостей об'єкта *objectName*.

Приклад

```
<script language = "JavaScript">
<! - -
function write_data ()
{
var k =0;
document.writeln ("<b>Об’єкт " + document + "</b><br>");
document.writeln ( "<table border = 1>" +
"<tr> <td>№</td><td align = middle> <b>" + Ім’я властивості +
"</ b> </ td> <td>значення властивості</td></ tr><br>");
for (i in document)
{
k++;
document.write ( "<tr> <td>" + k + </ td><td>" + i + "<br> </ td><td>" + i
+ "<br> </ td><br>");
}
document.write ( "</ tr> </ table>");
}
write_data ();
// -->
</script>
```

Щоб отримати список якорів, гіперпосилань, зображень необхідно скористатися об'єктами: anchor (anchors), link (links), image (images).

Об'єкт anchor (anchors) відповідає тегу <a name = " ...">. Всім міткам (якорям) відповідає масив anchors. Доступна властивість length, яка повертає кількість якорів у документі.

var Num_anch = document.anchors.length;

Объект link відповідає тегу <a href = “URL”> і тегу <area> в чутливих картах зображень:

<map name = “ім’я_обл”>

<area shape = “форма_обл” coords = “координати” href = “URL”>

Безлічі посилань відповідає масив links (тільки для читання).

Звернення до них: document.links[i] i = 0, 1, 2, …

document.links.length повертає кількість посилань у поточному документі.

Для звернення до властивості застосовується document.links[i].property name. Властивісті обєкту link:

- hash,
- host,
- hostname,
- href,
- pathname,
- port,
- protocol,
- search,

- target

Це практично всі властивості об'єкта location.

За допомогою цих двох об’єктів можна динамічно формувати переходи по мітках.

Об'єкт image відповідає малюнку в документі. Всі малюнки відповідають масиву images. Цій об’єкт дозволяє динамічно оновлювати зображення.

Є властивістю об'єкта document:

document. images[i]. property_name

Всі властивості відповідають атрибутам, крім властивості complete.

src - адреса зображення з високою роздільною здатністю.

lowsrc - адреса зображення з низьким дозволом.

Ці дві властивості можуть бути read/write.

height - висота зображення в пікселях;

width - ширина зображення в пікселях;

border - межа зображення;

vspace - вертикальний відступ зображення до кордону;

hspace - горизонтальний відступ зображення до кордону;.

Ці п'ять властивостей тільки only read.

Властивість complete приймає значення true/false і показує, завантажений малюнок в браузер чи ні.

Послідовність завантаження:

- малюється рамка;
- текст в атрибуті alt;
- образ по атрибуту lowsrc;
- образ по атрибуту src;

Якщо задані атрибути width і height, то сторінка завантажується значно швидше, так як браузер знає, де і як розміщувати картинки.

Приклад: Динамічна заміна зображення

<img src= “europe.gif” alt=“Europe”>

<form>

<input type= “button”

value=”Новий рисунок” onClick= “document.images[0].src= ‘Ukraine.gif’”>

</form>

Змінюємо карту Європи на карту України. Тут головна умова – розміри зображень повинні бути однаковими.

Література: [1], [4], [5], [6], [7].

Практичне заняття № 5.
Розробка JavaScript-програми для обробки подій

1. Створити програму, що запускає рядок, що біжить, у випадку, якщо курсор миші потрапив на текстове поле іншої форми, в якому при введенні не виявилося латинських літер.
2. Створити програму, що запускає рядок, що біжить, у випадку, якщо курсор миші потрапив на текстове поле іншої форми, в якому при введенні не виявилося букв кирилиці.
3. Створити програму виведення списку всіх гіперпосилань файлу різними шрифтами, що залежать від номера елемента у списку, якщо курсор миші потрапив на текстове поле форми, у якому при введенні цифр не виявилося.
4. Створити програму, що запускає годинник, що біжить, з датою в полі іншої форми, якщо курсор миші потрапив на текстове поле іншого кадру, в якому при введенні не виявилося великих літер кирилиці.
5. Створити програму формування розкривного меню зі списком для всіх форм файлу, якщо курсор миші потрапив на текстове поле, в якому при введенні не було великих латинських літер.
6. Створити програму формування розкривного списку всіх елементів форми файла, якщо курсор миші потрапив на текстове полі, у якому під час введення не виявилося малих латинських букв.
7. Створити програму формування розкривного меню зі списком всіх міток файлу, якщо курсор миші потрапив на текстове поле, в якому при введенні не було малих літер кирилиці.
8. Створити програму формування розкривного меню зі списком вмісту властивостей тієї гіперпосилання іншого форми, на яку потрапив курсор миші.
9. Створити програму виведення списку всіх форм файлу як горизонтальної таблиці різними шрифтами, який залежать від номера елемента у списку, якщо курсор миші потрапив на чисельне полі, у якому при введенні не виявилося букв.
10. Створити програму виведення списку всіх форм файлу у вигляді вертикальної таблиці різними шрифтами, що залежать від номера елемента у списку, якщо курсор миші потрапив на чисельне поле, у якому під час введення немає літер.
11. Створити програму виведення розкривного списку всіх властивостей того зображення, на яке потрапив курсор миші.
12. Створити програму виведення у вигляді розкривного списку інформації про скрипт (метод, host, повний шлях та ім'я скрипта) для всіх форм під час перехоплення відповідної кнопки Submit.
13. Створити програму, що запускає годинник, що біжить, з датою в однорядкове поле введення, якщо курсор миші пішов з текстового поля, в якому при введенні не виявилося великих літер кирилиці.

14. Створити програму видачі інформації (номер і текст) про вибрані елементи спадаючих списків іншого кадру з урахуванням атрибута MULTIPLE при виході курсору з текстового поля лише з латинськими літерами.

15. Створити програму видачі спадного списку назв усіх графічних зображень всіх форм поточного документа під час відходу курсору з деяким гіперпосиланням.

16. Створити програму, що дозволяє пересилати дані кожної форми на сервер у разі, якщо всі поля введення виявилися не порожні (щонайменше трьох форм із різною кількістю полів).

17. Створити програму формування посилань на всі каталоги шляху даного HTML-документа.

18. Створити програму виведення списку всіх форм як горизонтальної таблиці різними шрифтами, які залежать від номера елемента у списку, якщо курсор миші потрапив на поле, у якому під час введення не виявилося латинських букв.

19. Створити програму, яка запускає анімацію зображення з підрахунком кількості анімацій, коли курсор знаходиться на гіперзв'язку іншого кадру, і припиняє її, коли курсор йде з гіперзв'язку.

20. Створити програму виведення списку всіх гіперпосилань різними шрифтами, що залежать від номера елемента у списку, якщо курсор миші потрапив на текстове поле іншого кадру, в якому при введенні не було літер кирилиці.

21. Створити програму, що дозволяє пересилати дані кожної форми на сервер у разі, якщо всі поля введення містять текст довжиною не менше 10+N символів, де T – номер поля в формі (щонайменше трьох форм із різною кількістю полів).

Вказівки що до виконання заняття №5

Обробники подій

Для розширення можливостей HTML розроблені спеціальні атрибути - обробники подій. Користувач здійснює деяку дію, наприклад, клацає на гіперпосиланні або на кнопці *submit* для передачі даних. В цьому випадку генерується подія, яка перехоплюється інтерпретатором мови JavaScript, після чого проводиться відповідна обробка.

В основі всього цього механізму лежить форма HTML.

Основні події

click – натискання кнопки миші;

change – зміни в полях введення, багаторядковому полі, вибір елемента;

blur – зняття фокуса з елемента;

focus – задання фокусу на елементі;

load – завантаження сторінки;

unload – вихід з сторінки;
select - вибір елемента меню;
reset - скидання значень форми;
submit - передача даних форми;
mouseover - установка курсору миші на елементі;
mouseout - зняття курсору миші з елементу.

Додаткові події для малюнків

Abort – клацання миші на гіперпосиланні, натискання кнопки "*Stop*" або клавіші "*Esc*" при завантаження малюнка;

Error – помилка під час запуску малюнка.

Це може бути:

a) відсутній;
b) пошкоджений;
c) недоступний.

Load – для анімаційних файлів *GIF89a*. Виникає кожен раз, коли завантажується перший анімаційний кадр. При цьому завантаження окремих кадрів анімації виявити неможливо!

Нові події JavaScript

DbClick – подвійне натискання кнопки миші на гіперпосиланні або елементі форми**.**

DragDrop – перенесення об'єкта в вікно (наприклад, файл).

KeyDown – натискання клавіші;

KeyPress – натискання або утримання клавіші внизу;

KeyUp – відпустку клавіші;

MouseDown – натискання кнопки миші;

MouseUp – відпустку кнопки миші;

Move – користувач або сценарій рухає вікно;

Resize – користувач або сценарій змінює розмір вікна.

Приклад

```
<html> <head>
<script">
window.onresize = message;
// Тут ми задаємо процедуру message () обробки події resize,
// що виникає при зміні розміру вікна.
function message ()
{
   alert ("Розмір вікна змінено")
}
</script> </head>
        Змініть розмір вікна
<Body> </html>
```

Тут ми застосовуємо *window.onresize* так як об'єкт window не можна визначити через якийсь тег.

А для об'єктів типу *document* і нижчих можна було б через кнопки форм, не пов'язуючи з об'єктом *window*.

Одному тегу можуть відповідати кілька подій.

У наступному прикладі один клацання миші призводить до зміни кольору тексту, а подвійне клацання змінює колір фону тексту.

<p onClick = "this.style.color = 'blue'" ondblClick =
“this.style.backgroundColor = 'red'">
Приклад використання подій *onClick* і *ondblClick*.
</p>

Проходження подій

Розглянемо тему проходження подій по об'єктам, що входять в Об'єктну Модель Документа (DOM) і управління ними.

Група W3C, відштовхуючись від цього механізму в NN 4.x, де він з'явився, розробило свою концепцію, більш просту, яка зараз практично підтримується кожним браузером.

Приклад.

```
<html> <head>
<title> Проходження подій </title>
<style>
# obj
{ background – color: gray;
    left:       20;
        top:          20;
        width:       200;
        height:      100;
        position: absolute
}
# pic
{
  left:  30;
  top:  30;
  position: absolute
}
</style>
```

Тут вільно розташовується елемент, а в ньому вільно розташовується малюнок.

Фон елементу сторінки ми зробили сірим, щоб було помітніше.

Обробники події *onClick*, що виводять вікно попередження на екран.

```
<script language=”JavaSceipt”>
function picClick ()
{ window.alert (“Рисунок”)}
function objClick ( )
{window.alert (“Елемент сторінки”)}
```

function pageClick ( )
{ window.alert (“Сторінка”)}
</script> </head>
<body onClick = “pageClick ()”>
<div id = “obj” onClick = “objClick ()”>
<img src = “some.gif” id= “pic” onClick = “picClick ()”>
* *
</body> </html>

1. Клік на білому фоні - з'являється повідомлення про сторінку. Тут все звичайно.
2. Клік на сірому фоні (не по малюнку) - з'являються послідовно два повідомлення:

"Елемент сторінки
Після натискання "Ok"
"Сторінка"

Якби користувач клацнув два рази - спочатку на сірому тлі, а потім на білому.

Кожна подія передається вгору по об'єктної ієрархії. Тобто спочатку подія передається сірому елементу (що викликало цю подію), а потім - його батькові чи матері (тілу документа).

І кожен з цих об'єктів може обробляти ці події по різному.

3. Клік на малюнку - з'являються послідовно повідомлення:

- "Малюнок"

Після натискання "Ok"

- "Елемент сторінки"

Після натискання "Ok"

- "Сторінка"

Для скасування того, щоб повідомлення не проходило за ієрархії подій застосовують властивість cancelBubble.

За замовчуванням його значення false - що означає повне проходження подій. Для скасування повного проходження - true.

Наприклад,

function picClick ( )

```
{
alert (“Малюнок!”);
window.event.canselBubble= true;
}
```

Події клавіатури

Події keydown/keyup відбуваються при натисканні/відпуску клавіші і дозволяють отримати її скан-код у властивості keyCode. Скан-код клавіші однаковий в будь-якій розкладці і в будь-якому регістрі.

Наприклад, клавіша q може означати символ "q", "Q" або "й", "Й" в українській розкладці, але її скан-код буде завжди однаковий: 81.

Якими бувають скан-коди?

Для буквено-цифрових клавіш, скан-код буде дорівнювати коду відповідної великої англійської літери/цифри. Наприклад, при натисканні клавіші S (не важливо, який регістр і розкладка) її скан-код буде дорівнює "S".charCodeAt (0).

Подія keypress виникає відразу після keydown, якщо натиснута символьна клавіша, тобто натискання призводить до появи символу. Будь-які літери і цифри генерують keypress. Керуючі і функціональні клавіші, такі як Ctrl, Shift, F1, F2 .. - keypress не генерують.

Подія keypress виникає відразу після keydown, якщо натиснута символьна клавіша, тобто натискання призводить до появи символу. Будь-які літери і цифри генерують keypress. Керуючі і функціональні клавіші, такі як Ctrl, Shift, F1, F2 .. - keypress не генерують. Подія keypress дозволяє отримати код символу.

На відміну від скан-коду, він специфічний саме для символу і різний для "q" і "й".

Код символу зберігається у властивостях: charCode і which. Тут дуже велике число крос-браузерних несумісностей, які запам'ятати складно, тому на практиці потрібна лише одна «потрібна» функція, що дозволяє отримати код всюди.

Скасування призначеного для користувача введення

Поява символу можна запобігти, якщо скасувати дію браузера на keydown/keypress. Наприклад, якщо спробувати що-небудь ввести в цих полях:

```
<input onkeydown="return false" type="text" size="30">
<input onkeypress="return false" type="text" size="30">
```

то нічого не вийде.

В IE і Safari/Chrome скасування дії браузера при keydown також запобігає сама подія keypress. На момент спрацьовування keydown/ keypress клавіша ще не оброблена браузером. Тому в обробнику значення input.value - старе, тобто до введення.

А що, якщо необхідно обробити input.value саме після введення? Для цього необхідно - використовувати подію keyup, або запланувати обробітник через setTimeout (.., 0).

Скасування спеціальних дій

Скасовувати можна не тільки символ, а будь-яка дія клавіш. Наприклад, при скасуванні Backspace-символ не видалиться. А при скасуванні Page Down - сторінка не прокрутиться. Але існують дії, які в принципі не можна скасувати, в першу чергу - ті, які відбуваються на рівні операційної системи.

Комбінація Alt+F4 ініціює закриття браузера в Windows, що б ми не робили в JavaScript.

Автоповтор

При довгому натисненні клавіші виникає автоповтор. За стандартом, повинні генеруватися багаторазові події keydown (+ keypress), до того ж стояти властивість repeat = true у об'єкта події. Тобто, потік подій повинен бути такий:

keydown
keypress
keydown
keypress
..повтор, поки клавіша натиснута...
keyup

Однак в реальності на це покладатися не можна. Зараз під Firefox(Linux) генерується і keyup:

keydown
keypress
keyup
keydown
keypress
keyup
..повтор, поки клавіша натиснута...
keyup

У той же час Chrome під MacOS не генерує keypress. Покладатися можна тільки на keydown при кожному автонатисканні і keyup по відпускання клавіші.

Для роботи з тегом INPUT і рядом інших елементів, існують події форми.

Наприклад, oninput, яке спрацьовує після будь-якого введення в поле. Воно надійніше, так як відбувається в тому числі після вставки значення мишкою, взагалі без клавіатури. Розглянемо функцію runOnKeys(func, code_1, code_2, ... code_n), яка запускає func при одночасному натисканні клавіш зі сканкодами code_1, code_2, …, code_n.

Наприклад, код нижче виведе alert при одночасному натисканні клавіш "Q" і "W" (в будь-якому регістрі, в будь-якій розкладці).

```
runOnKeys (
function() { alert("Привіт!")},
"Q".charCodeAt(0),
"W".charCodeAt(0));
```

Алгоритм рішення:

- функція runOnKeys - зі змінним числом аргументів. Тому для їх отримання використовуємо властивість arguments;

- використовуємо два обробника: document.onkeydown і document.onkeyup. (Перший відзначає натискання клавіші в об'єкті pressed = {}, встановлюючи pressed [keyCode] = true, а другий - видаляє це властивість).

Якщо все клавіші з кодами з arguments натиснуті - запускаємо func.

- може виникнути проблема з повторним натисканням сполучення клавіш після alert.

```
function runOnKeys (func)
{
  var codes = [].slice.call(arguments, 1);
  var pressed = {};
  document.onkeydown = function(e)
  {
    e = e || window.event;
    pressed[e.keyCode] = true;
    for(var i=0; i<codes.length; i++)
    {  // перевірити, чи всі клавіші натиснуті
if (!pressed[codes[i]]) { return; }
     }
// під час показу alert, якщо відвідувач відпустить клавіші - не виникне
 // keyup при цьому JavaScript "пропустить" факт відпускання клавіш,
// a pressed [keyCode] залишиться true. Для уникнення "залипання" клавіші
// Обнуляємо статус всіх клавіш, нехай натискає все заново
     pressed = {};
  func(); };
document.onkeyup = function(e)
{
  e = e || window.event;
  delete pressed[e.keyCode];
};
}
```

Література: [5], [6], [7], [8]

Практичне заняття № 6.
Розробка JavaScript-програми із застосуванням шарів

Шари створюються із застосуванням тега **div** та каскадних таблиць стилів

1. Рисунок розбитий на горизонтальні та вертикальні прямокутники, частина з яких не видно. Після натискання кнопки поєднати плавно прямокутники в цілісний рисунок, спочатку по горизонталі, а потім - по вертикалі.
2. Рисунок розбитий на горизонтальні та вертикальні прямокутники, частина з яких не видно. По натисканні в будь-якому порядку двох кнопок, що відповідають за розташування смуг, плавно поєднати прямокутники у відповідному порядку в цілісний рисунок.
3. Коло розбите на три правильні колірні сектори. При попаданні курсору миші на будь-який сектор він повинен плавно розширюватися від центру до певного розміру або до моменту зняття курсора миші з сектора. На кнопці він повинен повертатися в початковий стан.
4. Коло розбито на три правильні колірні сектори. При попаданні курсору миші на будь-який сектор він повинен плавно стискатися до центру до моменту зняття курсору миші з сектора. На кнопці він повинен повертатися в початковий стан.
5. Квадрат розбитий на чотири колірні сектори. При попаданні курсору миші на будь-який сектор він повинен плавно розширюватися від центру до певного розміру або до моменту зняття курсора миші з сектора. На кнопці він повинен повертатися в початковий стан.
6. Квадрат розбитий на чотири колірні сектори. При попаданні курсору миші на будь-який сектор він повинен плавно стискатися до центру до моменту зняття курсору миші з сектора. На кнопці він повинен повертатися в початковий стан.
7. Коло розбите на шість колірних сектори. При попаданні курсору миші на будь-який сектор він повинен плавно розширюватися від центру до певного розміру або до моменту зняття курсора миші з сектора. На кнопці він повинен повертатися в початковий стан.
8. Коло розбите на шість колірних секторів. При попаданні курсору миші на будь-який сектор він повинен плавно стискатися до центру до моменту зняття курсору миші з сектора. На кнопці він повинен повертатися в початковий стан.
9. Створити у внутрішньому шарі плавне переміщення трьох іконок (невеликого розміру) випадково не виходячи за розміри екрана. По кнопці ховати чи відкривати цей шар.
10. Створити у внутрішньому шарі плавне переміщення трьох іконок (невеликого розміру) випадковим чином при попаданні курсору на одну з іконок, не виходячи за розміри екрана. По кнопці ховати чи відкривати цей шар.

11. Створити у внутрішньому шарі плавне переміщення однієї з трьох іконок (невеликого розміру) випадково при попаданні курсора миші на іконку, не виходячи за розміри екрана. По кнопці ховати чи відкривати цей шар.

12. Створити програму плавного обертання на різні боки чотирьох невеликих зображень навколо текстового центру.

13. Створити програму плавного обертання на різні боки чотирьох невеликих зображень навколо центрального зображення.

14. Створити програму плавного обертання на різні боки чотирьох текстових фрагментів навколо текстового центру.

15. Створити програму плавного обертання на різні боки чотирьох текстових фрагментів навколо центрального зображення.

16. Коло розбите на п'ять колірних секторів. При попаданні курсору миші на будь-який сектор він повинен плавно розширюватися від центру до певного розміру або до моменту зняття курсора миші з сектора. На кнопці він повинен повертатися в початковий стан.

17. Коло розбите на п'ять колірних секторів. При попаданні курсору миші на будь-який сектор він повинен плавно стискатися до центру до моменту зняття курсору миші з сектора. На кнопці він повинен повертатися в початковий стан.

18. Коло розбите на сім колірних секторів. При попаданні курсору миші на будь-який сектор він повинен плавно розширюватися від центру до певного розміру або до моменту зняття курсора миші з сектора. На кнопці він повинен повертатися в початковий стан.

19. Коло розбито на сім колірних секторів. При попаданні курсору миші на будь-який сектор він повинен плавно стискатися до центру до моменту зняття курсору миші з сектора. На кнопці він повинен повертатися в початковий стан.

20. Квадрат розбитий на чотири колірні сектори щодо осей. При попаданні курсору миші на будь-який сектор він повинен плавно розширюватися від центру до певного розміру або до моменту зняття курсора миші з сектора. На кнопці він повинен повертатися в початковий стан.

21. Квадрат розбито на чотири кольорові сектори щодо осей. При попаданні курсора миші на будь-який сектор, він повинен поступово стискатися до центру до моменту зняття курсора миші з сектора. По кнопці він повинен повертатися в початковий стан.

22. Створити програму обертання в різні боки п'яти невеликих зображень навкруги текстового центру.

23. Створити програму обертання в різні боки п'яти невеликих зображень навкруги центрального зображення.

24. Створити програму обертання в різні боки п'яти текстових фрагментів навкруги текстового центру.

25. Створити програму обертання в різні боки п'яти текстових фрагментів навкруги центрального зображення.

Вказівки що до виконання заняття №6
Вкладені шари

A. **Сторінка з перетягуванням елементів**

Як приклад розглянемо такі деталі:

1. Кілька елементів для перетягування.
2. Відображення координат курсора миші в рядку статусу.

```
<html> <head>
<title> перетягування елементів
</title>
<style>
div {background–color: red; position: absolute}
</style>
```

Для того, щоб не ставити одне і теж для елементів.

```
<script language=“JavaScript”>
function bodyMouseMove ()
{       var obj;
window.status =
“X: ”+ window.event. clientX +
“Y: ”+window.event.clientY;
// Тут показуються координати миші в рядку стану.
obj = window.event.srcElement;
        if (obj.tagName != “BODY”&& window.event.button == 1)
// Якщо подія була передана вперше не тілу документа і натиснута ліва
// кнопка миші, то виконуємо перетягування
  {
        obj.style.pixelLeft = window.event.x – obj.style.pixelWidth/2;
        obj.style.pixelTop = window.event.y – obj.style.pixelHeigth/2;
  }
}
</script>
```

Визначаємо координати по X та Y.

Позиціонуємо елемент так, щоб його центр співпадав з курсором миші.

```
</head>
<body onMouseMove = “bodyMouseMove ( );”>
```

Перетягування елементів в сторінці реалізує обробник події *onMouseMove*.

Тут використовується проходження подій. Навіть якщо подія буде передано елементу, згодом воно в будь-якому випадку потрапить до того, яке його і обробляє.

Далі будуємо чотири елементи сторінки, які ми будемо перетягувати з місця на місце. Для кожного елемента визначаємо стиль - положення прямокутника. Кольори задаються по успадкування з таблиці стилів.

```
<div id = “div1” style = “left: 20; top: 20;
```

```
        width: 30 height: 40”>
    </div>
    <div id = “div2” style =“left: 120 top: 20; width: 30 height: 40”>
    </div>
    <div id = “div3” style =“left: 20 top: 120; width: 30 height: 40”>
    </div>
    <div id = “div4” style =“left: 120     top: 120; width: 30 height:      40”>
    </div>
    </body> </html>
```

В. **Сторінка з рисунком-покажчиком**

```
    <html> <head>
    <title> Курсор з рисунком-покажчиком
</title>
    <style>
     img {position: absolute}
    </style>
```

Робимо один елемент з вільним позиціонуванням.

```
    <script language = “JavaScript”>
    var time_id, x, y, ix, iy;
    // Задаємо глобальні змінні таймера і координат.
    function setupAnim ()
    {
     time_id = window.setInterval (
        “imgMove ()”, 50);
    }
    // Ця функція виконується при завантаженні сторінки і
    // створює таймер (інтервал виклику функції)
    function imgMove ()
    // Рух малюнка-покажчика проводиться невеликими збільшеннями
    // по 4 пікселя (або менше).
    // (x,y) – положення курсору
    // (ix,iy) – положення покажчика малюнка
    {if (x >= ix)
            {
               if (x – ix > 4)
          ix = ix + 4
      else
               ix = x
            }
     else
            {
          if (ix-x > 4)
```

```
        ix = ix - 4
      else
        ix = x
      }
    if (y >= iy)
    {  if (y-iy > 4)
          iy  = iy  + 4
    else
      iy = y;
    }
    else
    {
      if (iy-y > 4)
        iy = iy - 4
      else
        iy=y;
     }
```

// Цей фрагмент коду обчислює приріст руху,

// погодившись з напрямком руху курсора миші.

```
    pImage.style.pixelLeft =
    ix – pImage.style.pixelWidth;
    pImage.style.pixelTop = iy;
```

// Тут ми встановлюємо нові координати рисунка-покажчика таким чином,

// щоб на місці курсору перебував лівий верхній кут рисунка.

```
    function bodyMoveMouse( )
    {
      x = window.event.clientX;
      y = window.event.clientY;
    }
```

// Зберігаємо координати курсора миші для використання в процедурі руху

// рисунка, так як ми вже не будемо мати доступ до об'єкту event.

```
    </scrint> </head>
    <body onload = “setupAnim ();” onMouseMove = “bodyMoveMouse;” >
    <img src = “my.gif” id = “pImage” style = “top:0 left: 0 width: 30”>
    Первісне значення рисунка-покажчика.
    </body> </html>
```

C. **Обертання 2-х текстів навколо центрального рисунка**

1. Елемент CSS (таблиця стилів).
2. Можливість браузера працювати з DOM (об'єктна модель документа).

```
    <html> <head>
    <title>. . . . . .</title>
```

Таблиця стилів полягає в теги *<style>…</style>*.

Цей тег *<style>* може містити необхідний атрибут *type* що містить обов'язкове значення *text/css*.

```
<style type = “text/css”>
.fly {  font-size: 24 px;
            position:absolute;
            visibility: hidden;
            z-index: 2
      }
```

font-size задає розмір шрифту в пікселях.

position – для розміщення елемента (відносно чого координати будуть зчитуватися):

а) *absolute* – відносно верхнього лівого батьківського елемента;

б) *relative* – відносно позиції;

в) *static* – за замовчуванням, тобто без вільної позиції.

```
body
{ font-family: arial;
  background: #006000;
 }
```

font-family – задає шрифт для тексту. Може бути кілька шрифтів через кому;

background – колір фону

```
</style>
<script type= “text/javascript”>
```

// сучасний стандарт підключення скрипта на JavaScript в HTML-сторінку.

```
function rot ( )
{
  for(var i=0; I<pos.length; i++)
        {
          pos[i]+=inc;
          objects[i].visibility=’visible’;
objects[i].left=(r*Math.cos(pos[i]))+xoff;

objects[i].top=(r*Math.sin (pos[i]))+yoff;
setTimer = setTimeout (“rot( )”,50);
        }
}
// закінчення функції rot ()
fuction initObj ()
// Для ініціалізації шарів
{
  objects = new Array(fly1,fly2);
```

```
// Оголошуємо масив шарів, назвавши їх fly1 і fly2.
pos = new Array (0);
pos[0] = 0;
for (var i=1; i<objects.length; i++)
//  Позиція наступного шару щодо попереднього.
{
        pos[i] = parseFloat (pos [i-1] + ((2*pi) / objects.length));
// Позиціонування наступного шару щодо попереднього
}
rot();
} закінчення initObj ()
// Змінні для обертання
var objects;
var pos;
var r=200; // радіус
var xoff = 220; // зсув по x
var yoff =220;  // зсув по y
var pi = Math.PI;
var inc = pi/180; // крок зсуву при обертанні
</script>
</head>
<body>
<h2> Обертання 2-х текстових фрагментів навколо центрального малюнка </h2>
<p id = “fly1 class = “fly”> текст_1 </p>
<p id = “fly2 class = “fly”> текст_2 </p>
```

id – аналог атрибута *name*;

class – вказівка, яку частину таблиці стилів буде використана під ім'ям *fly*.

```
<p> <img src = “1.gif” hspace = “130” vspace = “80” z-index = “1”> </p>
<script type = “text”/JavaScript”>
// Визначаємо змінні, які є покажчиками на шари
var fly1 = document. all. fly1.style;
var fly2 = document. all. fly2.style;
initObj ( );
// Викликаємо функцію ініціалізації шарів.
//      За її завершення обертання шарів почнеться автоматично.
</script>
</head>
<body>
<div style = “top:80; left:130
position: absolute”>
<img src = “1.gif”>
</div>
```

</body>
</html>

Вільно позиціонувати можна тільки блокові елементи і позиції списку.

У прикладі ми позиціонували рядок тексту, укладеного в теги абзацу <p ... . Для звернення ми теж застосували тег , тобто помістили в блоковий тег. Але краще поміщати його в парні теги ... . Візуально тег нічого не робить. Він просто групує один або кілька елементів в сторінці в один для того, щоб надалі їм можна було управляти як єдиним цілим блоком.

Література: [1], [3], [4], [5], [6]

Практичне заняття №7
Розробка додаткових методів для об'єктів JavaScript

У цій роботі необхідно створити функцію, яка виконуватиме задану дію, після чого за допомогою властивості prototype ця функція пов'язується з методом.

Для позначення об'єкта у функції, з яким вона буде застосовуватися, застосовується оператор this.

Створений спосіб перевірити на трьох елементах об'єкта. Елементи об'єкта мають бути під різними іменами.

1. Прибрати заданий елемент у масиві.
2. Прибрати кілька заданих елементів у масиві.
3. Прибрати кілька перших елементів у масиві.
4. Прибрати кілька останніх елементів у масиві.
5. Вставити після заданого елемента у масиві кілька нових елементів.
6. Замінити кілька перших елементів масиву.
7. Замінити кілька останніх елементів масиву.
8. Реверсувати розташування елементів у масиві.
9. Реверсувати символи у рядку.
10. Реверсувати символи кожного слова у рядку.
11. Прибрати початкові прогалини у рядку.
12. Прибрати кінцеві прогалини у рядку.
13. Сформувати масив слів, що містять заданий символ у рядку.
14. Сформувати масив слів, що містять задану послідовність символів у рядку.
15. Сформувати зі спискового масиву асоціативний масив.
16. Сформувати асоціативний масив слів із рядка.
17. Поміняти місцями в асоціативному масиві ключі та значення.
18. Знайти у двох масивах різні елементи і сформувати їх новий масив.
19. Збільшити розмір масиву до заданої довжини заданим значенням.
20. Вибрати кілька випадкових елементів із масиву.
21. Знайти елементи асоціативного масиву, що збігаються із заданим значенням, та сформувати список їх ключів.

22. Сформувати перелік індексів унікальних елементів масиву.

23. Сформувати перелік індексів неунікальних елементів масиву.

24. Сформувати перелік ключів унікальних елементів асоціативного масиву.

25. Сформувати перелік ключів неунікальних елементів асоціативного масиву.

26. Прибрати останній елемент масиву з необхідним значенням.

Вказівки що до виконання заняття №7

Всі об'єкти мають властивість prototype для створення нових конструкторів об'єкта.

Нехай *countChars()* – функція підрахунку появи символу в рядку, що знаходиться в аргументі.

Тоді створення нового методу відбувається за наступним синтаксисом:

String. prototype.count = countChars;

count – новий метод, що стане конструктором об'єкта *String*;

countChars – посилання на функцію *countChars()*.

Після цього новий метод *count* може бути застосований до будь-якого об'єкта типу *String*.

Приклад:

```
function countChars (arg)
{
  var hits  = 0;
  pos = this.toLowerCase().indexOf(arg);
 while ( pos != -1 )
        {  hits ++;
           pos = this.toLowerCase().indexOf(arg,pos+1);
        }
 return(hits);
}
String. prototype.count = countChars;
```

// Прив'язка функції countChars (arg) до нового методу *count* об'єкту *String*.

```
var myStr = new String("Деякий рядок нового тексту");
var numChar = myStr.count("o");
alert( "Тут символ 'o' зустрічається" + numChar + "раз" );
```

або таки виклики:

```
var numChar ="Деякий рядок нового тексту".count("o");
var newStr ="Додавання нових методів-конструкторів в об'єкт";
var numChar = newStr.count("i");
```

Оператор *this* всередині функції посилається на той об'єкт, що буде йти з методом *count*.

У першом прикладі - змінної *numChar* присвоєно значення виразу *myStr.count("o")*.

Визначення методу *count()* за допомогою властивості prototype прив'язує функцію *countChars ()*.

Тобто при зверненні до методу *count ()* викликається функція *countChars()*.

Функція – метод-конструктор додається тільки через властивість ***prototype***, що при зверненні до доданого конструктору згадувати не треба.

Література: [1], [5], [6], [7]

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Davis, Ashley. Data Wrangling with JavaScript / Ashley Davis. - 2019. - 433 p.
2. Pro HTML5 Programming : Powerful APIs for Richer Internet Application Development / Peter Lubbers, Brian Albers, Frank Salim. - Apress, 2011. – 352 p.
3. Sawyer, David. McFarland. CSS3 / David Sawyer. - 4-th ed. - O'Reilly Media, 2015. - 718 p.
4. Robson, Elisabeth. Head First HTML and CSS / Elisabeth Robson, Eric Freeman. - 2-nd ed.- Reilly Media , 2012. - 764 p.
5. Flanagan, David. JavaScript : The Definitive Guide / David Flanagan. - 6-th yd. - O'Reilly Media, 2011. – 1098 p.
6. Williams, Gordon F. Making Things Smart: Easy Embedded JavaScript Programming for Making Everyday Objects into Intelligent Machines / Gordon F. Williams. - Maker Media, 2017. – 352 p.
7. Лазарович, І.М. Конспект лекцій з дисципліни «Програмування та підтримка веб-застосувань» для студентів напряму підготовки «Інформатика» / І.М. Лазарович. – Івано-Франківськ : вид-во Прикарпат. Нац. ун-ту імені Василя Стефаника, 2015. - 153 с.

НАВЧАЛЬНО−МЕТОДИЧНЕ ВИДАННЯ

Методичні вказівки до виконання практичних занять з дисципліни «Основи Web-програмування» для студентів ОС «бакалавр» денної та заочної форм навчання галузі знань 12 Інформаційні технології спеціальності 122 Комп’ютерні науки [Електронний ресурс] / укл. В.І. Костін. – Луцьк : ДонНТУ, 2023. – 45 с.

.

Комп’ютерний набір і верстка: Костін В.І.

Укладач: старший викладач кафедри ПМІ Костін В.І.

ДВНЗ «Донецький національний технічний університет»
43018, м. Луцьк, вул.Потебні, 56.