МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ

до практичних занять
з дисципліни «WEB-ТЕХНОЛОГІІ ТА WEB-ДИЗАЙН»

для студентів усіх форм навчання
спеціальності 122 «Комп'ютерні науки»

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою ІУС.
Протокол № 11
від 13.04.21 р.

ХАРКІВ 2023

Методичні вказівки до практичних занять з дисципліни «Web-технології та web-дизайн» для студентів спеціальності 122 «Комп’ютерні науки», [Електронний ресурс] / упоряд.: Т.І. Борисенко. – Електронне видання. – Харків: ХНУРЕ, 2023. – 50 с. – pdf

Упорядники Т.І. Борисенко

Рецензент: Н.Г. Аксак, д.т.н., проф. каф. КІТС

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ДСТУ – державний стандарт України
ІС – інформаційна система
ПЗ – практичне заняття
CSS – Cascading Style Sheets
HTML– HyperText Markup Language
UML– Unified Modeling Language

ЗМІСТ

ВСТУП

Дисципліна "Web-технології та web-дизайн" є однією з професійно-орієнтованих дисциплін, яка викладається в рамках базової підготовки бакалаврів з напряму "Комп'ютерні науки".

Курс призначений для студентів другого курсу, його вивчення базується на знаннях, отриманих при вивченні дисциплін "Алгоритмізація та програмування", "Операційні системи та системне програмування", "Об'єктно-орієнтоване програмування".

Тематика лекційних занять і деяких тем, які надаються для самостійного вивчення, зорієнтована на засвоєння теоретичних основ дисципліни "Web-технології та JavaScript". Їх вивчення допомагає добре підготуватися до виконання практичних занять.

У запропонованих методичних вказівках наведено п'ять практичних занять з дисципліни "Web-технології та JavaScript", послідовне виконання яких спрямоване на придбання практичних навичок роботи з сучасними засобами створення Web-документів.

1 РОБОТА З ТЕКСТОМ, ЗОБРАЖЕННЯМИ І ТАБЛИЦЯМИ ЗАСОБАМИ HTML ТА CSS

1.1 Мета заняття

Мета заняття полягає в ознайомленні з базовими HTML-тегами та правилами їх застосування, з властивостями CSS для форматування та налаштування блоків і в набутті практичних навичок роботи з ними.

1.2 Методичні вказівки з організації самостійної роботи студентів

Готуючись до практичних занять, необхідно повторити лекційний матеріал та матеріал для самостійної роботи, який стосується основ HTML і словник елементів HTML та атрибутів [1, 2, 3].

1.3 Теоретичні положення

1.3.1 Види сайтів

Сайти створюються з різними цілями: щоб продавати товари, просувати послуги, розкручувати бренд, надавати інформацію і т.д.

Сайт-візитка зазвичай має унікальний, але простий і функціональний дизайн. Підходить для компаній, які хочуть розмістити інформацію про себе і свої послуги в Інтернеті. Такий сайт, як правило, містить такі розділи: "Про компанії", "Продукція або послуги", "Прайс-листи", "Контактна інформація".

Інтернет магазин – це online-каталог продукції, з якого можна прямо на сайті купити потрібні товари або послуги, оплатити їх та замовивити доставку. Можливість зробити online-замовлення – це обов'язкова ознака магазину. Ця можливість доповнюється функцією кошика, яка дозволяє за один раз замовити кілька товарів або скільки завгодно послуг.

Промо-сайт – це сайт, який створюється спеціально для просування будь-якого товару (групи товарів) або послуги. Він являє собою рекламний інструмент. Основне завдання промо-сайту – надання користувачам найсвіжішої та актуальної інформації про товари або послуги. Інформація повинна переконати потенційного клієнта в тому, що надаються компанією товари і послуги є найкращими серед подібних на ринку.

Інформаційний сайт представляє собою досить великий віртуальний масив інформації, що включає в себе безліч різних тематичних розділів, або кілька самостійних проектів. Він є для клієнта основним джерелом інформації, нагадує енциклопедію або спеціалізований журнал.

Інформаційний портал – це великий ресурс, що належить компанії з широким колом видів діяльності. Інтернет-портал оглядає велику кількість тем. Його відмінною рисою є наявність великої кількості і широкого діапазону різних сервісів для користувачів: пошта, голосування, форум, особистий кабінет, чат тощо.

Корпоративний сайт – це фірмовий комерційний сайт компанії. Основним завданням корпоративного представництва в мережі є просування бізнесу компанії і автоматизація її діяльності. Корпоративний сайт включає в себе одночасно елементи інтернет-магазину, сайту-візитки і інформаційного ресурсу.

За допомогою корпоративного сайту можна значно підвищити впізнаваність бренду. Корпоративний сайт також автоматизує діяльність компанії і допомагає вести бухгалтерський облік. У нього включена підтримка користувачів в режимі онлайн. Реклама на корпоративному сайті залежить від цілей і завдань компанії. На такому сайті є власна система пошуку по сайту, можливість каталогізації, продуманий зв'язок з користувачами.

Електронні торгові майданчики об'єднують в єдиному інформаційному просторі постачальників і споживачів різних товарів / послуг, надаючи учасникам торгових угод сервіси, що підвищують ефективність процесів. На торговій онлайн-майданчику будь-який зареєстрований в системі користувач зможе продати що-небудь або придбати потрібну йому річ.

1.3.2 Призначення і уміст головної сторінки сайту

Будь-який сайт завантажується з головної (домашньої) сторінки (home page), яка, перш за все, повинна давати відповідь на питання: "Що це за сайт (система)?".

На головній сторінці відвідувач повинен отримати відповіді на такі питання:

- "Хто ви?"
- "Чим займаєтеся?"
- "Чим ви можете мені допомогти?"
- "Чому я повинен вам вірити?"

- "Є докази вашої хорошої роботи?"
- "Гарантії?"
- "Адреса? Телефон? Ваші контакти?"

Відповіді на ці питання не обов'язково давати на головній сторінці. Можна на видному місці в меню розмістити посилання, які перенаправляють користувача на сторінку з потрібною інформацією. Зазвичай це сторінки "Про нас", "Гарантії", "Контакти" і т. ін.

На головній сторінці сайту можуть розташовуватися такі елементи:

- назва сайту;
- логотип;
- навігаційний блок (посилання на основні розділи сайту);
- контакти;
- вхід в особистий кабінет користувача (якщо є);
- пошук по сайту;
- кошик (якщо є);
- відгуки;
- портфоліо (якщо є);
- текст з описом змісту сайту;
- кнопка заклику до дії.

Логотип – це фірмовий знак, який люди асоціюють з конкретним брендом. Логотип – це перше знайомство клієнта з компанією. Саме по ньому багато хто судить про пропоновані послуги і товари.

Логотип зазвичай розташовується в лівому верхньому кутку. він повинен бути простим і таким, що запам'ятовується. Це може бути якийсь символ, незвично оформлена назва компанії або навіть розпис. До логотипа потрібно пояснення.

Контакти можуть бути винесені на окрему сторінку, але тоді один з телефонів або електронну пошту необхідно вказати в шапці і футере сайту.

Кнопка пошуку по сайту потрібна на багатьох сайтах, але особливо в інтернет-магазинах. Зазвичай її розташовують праворуч у верхній частині сторінки або посередині.

У тексті з описом змісту сайту може бути зазначено, чим займається компанія і скільки часу, як багато було виконано робіт. Текст може містити посилання на портфоліо, де можна ознайомитися з роботами. Текст так само може висвітлювати тему сервісу і гарантій.

Портфоліо на головній сторінці може містити кілька робіт і посилання на інші роботи.

Відгуки підвищують рівень довіри нових клієнтів до компанії. Необхідно опублікувати хоча б 3-4 відгуку.

Кнопка заклику до дії не завжди потрібна на головній сторінці. Якщо сайт – це інтернет-магазин, то замість заклику до дії краще зробити блок з найпопулярнішими товарами. У цьому випадку покупець зацікавиться і перейде на сторінку за детальною інформацією. А ось на ній кнопка заклику до дії буде вже доречна.

Якщо передбачений особистий кабінет користувача, то на головній сторінці повинен бути розташований вхід в нього (зазвичай справа вгорі) і корзина (для інтернет-магазинів) трохи нижче входу в особистий кабінет або на одному з ним рівні.

У нижній частині сайту (footer) можна продублювати контакти і верхнє меню. Тут же можна дати посилання на Політику конфіденційності, Програму лояльності, FAQ (якщо він передбачений).

1.3.3 Настрій, що викликається кольором

Сучасні Web-дизайнери намагаються використовувати вплив кольору на емоції людини, щоб створювати вірну атмосферу для кожного сайту. На основі декількох досліджень розглянемо, як кольори впливають на емоції та допомагають створювати UX-дизайн.

Різні культури по всьому світу по-різному сприймають кольори. Опишемо емоційні асоціації, характерні тільки для західної культури.

Червоний колір асоціюється з владою, важливістю, молодістю. Це найбільш стимулюючий колір, настільки енергетично заряджений, що може навіть збільшити кров'яний тиск.

Червоний колір представляє пристрасть і силу, він більше за інші кольори привертає увагу, тому часто використовується для важливих попереджень та оголошень. Проте люди прагнуть швидше покинути «небезпечну зону» червоного кольору та прокручують сторінку донизу, що, у свою чергу, дозволяє показати користувачеві більше контенту.

Але червоний колір може спрацювати і негативно, оскільки може викликати агресію або надстимуляцію. Якщо потрібно створити більш розслаблюючу атмосферу, намагайтеся використовувати червоний помірно та вибирайте більш світлі відтінки червоного.

Жовтогарячий колір асоціюється з дружелюбністю, енергією, унікальністю. Він найспокійніший з теплих тонів і здатний викликати цілий

спектр різнобічних емоцій. В якості основного кольору він може викликати інтерес і бадьорити, а в якості вторинного – зберігати ці властивості, але у більш ненав'язливій манері. Крім того, жовтогарячий допомагає створити відчуття руху та енергії. Колір асоціюється з творчістю, при цьому зберігається відчуття знайомого бренду.

Жовтий колір асоціюється з щастем, ентузіазмом та архаїчністю (більш темні тони). Це один з найбільш універсальних кольорів, а емоції, які він викликає, здебільшого залежать від відтінку.

Яскравий жовтий колір додає енергії, але без гостроти та різкості, яка присутня у червоному. Середні відтінки жовтого кольору викликають відчуття комфорту, хоча все ще бадьорять. Темні відтінки (включаючи золотий) забезпечують відчуття старовини, наповнюють простір мудрістю і цікавістю.

Зелений колір асоціюється з зростанням, стабільністю, темою фінансів, темою навколишнього середовища. Він поєднує теплі та холодні відтінки, хоча більше схиляється до холодних. Це означає, що зелений має розслаблюючу дію синього кольору, але також має трохи енергійності жовтого. У результаті цього він створює дуже збалансовану і стабільну атмосферу. Більш темні відтінки зеленого створюють враження багатства і достатку.

Синій колір асоціюється зі спокоєм, безпекою, відкритістю (більш світлі відтінки), надійністю (більше темні відтінки). Як і у випадку з жовтим кольором, вплив синього сильно залежить від відтінку. Всі відтінки синього універсальні щодо розслаблення і безпеки, але світлі тони асоціюються з дружелюбністю, а темні – з сумом.

Фіолетовий колір асоціюється з розкішшю, романтикою (світлі відтінки), містикою та таємницею (темні відтінки). Пурпурні відтінки відображають щедрість і багатство загалом, що робить їх відмінним вибором для модних товарів і предметів розкоші або, навіть для шоколаду. Більш світлі відтінки, такі як лавандовий (фіолетовий з додаванням рожевого), навівають думки про романтику, тоді як темні відтінки здаються більш шикарними і таємничими.

Чорний колір асоціюється з владою, вишуканістю, нервозністю. Він може викликати різні асоціації залежно від супроводжуючих його кольорів або домінувати над ними, якщо використовувати його надмірно.

Сила і нейтральність чорного роблять його відмінним вибором для великих блоків тексту, але як основний колір він може створити відчуття нервозності або навіть асоціюватися зі злом. Для більшості сайтів чорний використовується, щоб створити відчуття вишуканості. Від поєднання чорного і білого в мінімалістичному дизайні створюється враження елегантності і стилю.

Білий колір асоціюється з чистотою, простотою, чеснотою і невинністю. Цей колір часто використовують для фону мінімалістичних і простих сайтів.

Крім того, жоден колір не дозволить звернути стільки уваги на інші кольори, як білий.

Сірий колір асоціюється з нейтральністю, формальністю, меланхолією. У деяких ситуаціях сірий може створювати похмуру і сумну атмосферу, але він все ж часто використовується професійними дизайнерами. Вся справа у відтінках: чергуючи їх, можна отримати всі емоції, викликані як чорним кольором, так і білим. А у поєднанні з більш яскравими кольорами в дизайні сірий фон здається сучасним, а не похмурим.

Бежевий колір передає характер інших кольорів. Сам по собі він досить тьмяний і невиразний, але у нього є одна чудова властивість: бежевий приймає характер кольорів, які його оточують. Тому, якщо він призначений не для видображення стриманості, бежевий служить у якості фону. Більш темні відтінки бежевого створюють відчуття традиційності і приземленості, дають відчуття паперової текстури, а більш світлі відтінки здаються сучасними.

Колір слонової кістки асоціюється з комфортом, елегантністю, простотою. Колір слонової кістки, а також кремовий викликають практично ті ж емоції, що і білий. Проте колір слонової кістки більш теплий, ніж білий, що створює більше відчуття комфорту, зберігаючи при цьому мінімалізм. Можна використовувати колір слонової кістки замість білого, щоб пом'якшити контраст між ним і темнішими кольорами.

1.4 Виконання завдання

1.4.1 Отримайте у викладача індивідуальне завдання.

1.4.2 Розробіть структуру макета головної HTML-сторінки для веб-сайту, зазначеного в індивідуальному завданні. Поділіть її на зони. Схему макета зобразіть у вигляді малюнка в Word або в іншому доступному редакторі. Для кожної зони вкажіть її уміст, якщо її найменування не розкриває його суті. Приклад схеми, що відображає структуру сторінки і вміст блоків, наведено на рис.1.1.

1.4.3 Головна сторінка може включати різні елементи і блоки, в залежності від спрямованості сайту, але обов'язковими є такі:

- назва сайту;
- навігаційний блок (реалізовувати його поки ще не потрібно – достатньо залишити для нього місце на сторінці);
- заголовки;

- інформація, яка представлена в табличному вигляді (її можна розмістити на окремій сторінці, перехід на яку повинен здійснюватися за посиланням з головної сторінки);
- зображення (з пояснювальним текстом).

1.4.4 Визначте, який матеріал слід розмістити на сайті. Підготуйте необхідні тексти і файли зображень, відео, анімації.

Рисунок 1.1 – Приклад структури макета головної сторінки

1.4.5 Виберіть для сайту колірну схему, а потім за допомогою одного з online-сервісів (*color.adobe.com, colorhunt.co, coolors.co)* виберіть кольори для сайту і визначте призначення для кожного кольору.

1.4.6 Розробіть код головної HTML-сторінки. При створенні блоків, по можливості, використовуйте елементи, що приписують семантику, відповідну змісту блоку.

1.4.7 Основний контент сторінки розмістіть обов'язково у кілька колонок.

1.4.8 Головна сторінка повинна містити хоча б одне зображення.

1.4.9 Продемонструйте розроблені сторінки викладачеві в браузері.

1.4.10 Складіть звіт, оформіть його відповідно до ДСТУ 3008: 2015 [4] і захистити роботу.

1.4.11 Звіт повинен міститиме:

- номер індивідуального завдання і найменування розроблюваного сайту;
- схему структури макета головної сторінки сайту;
- назву обраної колірної схеми і перелік обраних кольорів для сайту з описом їх призначення;
- HTML-код всіх розроблених сторінок;
- CSS-код, якщо він знаходиться в окремому файлі;
- скріншоти вікон браузера з розробленими веб-сторінками, які демонструють виконання завдань до практичного заняття.

1.5 Контрольні запитання

1. Що зазвичай містить головна сторінка сайту?
2. Які види колірних схем розрізняють? Яке їхнє призначення?
3. Чим відрізняються блокові елементи HTML від рядкових? Наведіть приклад рядкового елемента HTML.
4. Що може містити контейнерний тег <head>?
5. В якому тегу HTML вказують ключові слова веб-сторінки?
6. Для чого служить тег <meta>?
7. Які теги не може включати в себе тег <p>?
8. Яке призначення тега ?
9. Як реалізуються списки визначень в HTML і як вони відображаються в браузері?
10. Чим тег <h1> відрізняється від тега <h6>?
11. Чим відрізняються теги <section>, <article> від тега ?
12. Як можна позиціонувати елемент сторінки?
13. З чого складаються значення властивостей блоку width і height?
14. Де знаходиться область padding в блоці?
15. При якому виду позиціонування елемент зміщується щодо свого початкового положення?

16. Як можна управляти накладенням елементів один на одного при позиціонуванні?

17. Які існують способи додавання таблиць стилів на сторінку? Який метод підключення таблиць стилів забезпечує найвищій пріоритет впливу стилів на елемент?

18. Для чого служать селектори в правилах стилів?

19. Як працюють селектори атрибутів?

20. Які частини може мати таблиця в HTML?

21. Чим формат файлу зображення PNG-24 відрізняється від формату JPEG?

22. З якою метою створюють групи колонок в таблиці?

23. Як реалізувати вкладення однієї таблиці в іншу?

24. Як реалізується об'єднання клітинок таблиці в HTML?

2 СТВОРЕННЯ МЕНЮ САЙТУ

2.1 Мета заняття

Мета заняття полягає в ознайомленні з підходами до створення навігаційних блоків сайту і набутті практичних навичок зі створення багаторівневого меню сайту.

2.2 Методичні вказівки з організації самостійної роботи студентів

Готуючись до заняття, необхідно повторити лекційний матеріал, який висвітлює використання псевдокласів, властивості CSS для форматування блоків, обтікання і позиціонування та особливості використання модуля макета гнучкого контейнера Flexbox.

2.3 Теоретичні положення

2.3.1 Створення горизонтального меню

Один із способів створення горизонтального меню – це стилізація звичайного списку. Створимо звичайний маркований список:

```
<ul id="navbar">
        <li><a href="#">Головна</a></li>
        <li><a href="#">Новини</a></li>
        <li><a href="#">Контакти</a></li>
        <li><a href="#">Про нас</a></li>
</ul>
```

Далі необхідно скинути стиль, використовуваний для списків за замовчуванням. Крім того, щоб пункти списку розташовувалися один за одним по горизонталі, необхідно перевизначити їх з блокових на рядкові:

```
#navbar {
        margin: 0;
        padding: 0;
        list-style-type: none;  /* Видалення маркерів / нумерації списку */
}
#navbar li { display: inline; }
```

Тепер визначимо стильове оформлення для горизонтального меню, наприклад, наступним чином:

```
#navbar {
        margin: 0;
        padding: 0;
        list-style-type: none;
        border: 2px solid #0066FF;
        border-radius: 20px 5px;
        width: 550px;
        text-align: center;
        background-color: #33ADFF;
        }
#navbar li { display: inline; }
#navbar a {
        color: #fff;
        padding: 5px 10px;
        text-decoration: none;
        font-weight: bold;     /* Ступінь товщини шрифту */
        display: inline-block;   /* Елемент рядковий, але при цьому може мати
                                    властивості блоку */
        width: 100px;
}
#navbar a:hover {
        border-radius: 20px 5px;
        background-color: #0066FF;
}
```

2.3.2 Створення меню, що випадає

Для створення меню, що випадає, його підпункти необхідно розмістити в окремому списку і вкласти його в елемент <*li*>, який містить батьківське посилання щодо підпунктів:

```
<ul id="navbar">
        <li><a href="#">Головна</a></li>
        <li><a href="#">Новини</a></li>
        <li><a href="#">Контакти</a>
        <ul>
```

```
            <li><a href="#">Адреса</a></li>
            <li><a href="#">Телефон</a></li>
            <li><a href="#">Email</a></li>
        </ul>
        </li>
        <li><a href="#">Про нас</a></li>
</ul>
```

Щоб список з підпунктами не відображався на веб-сторінці весь час, його необхідно приховати за допомогою такої властивості:

display: none;.

Зробимо так, щоб список з підпунктами відображався при наведенні курсору на батьківський елемент *<li>*. Для цього вкладений список підпунктів повинен бути знову перетворений в блоковий елемент. Використовуємо для цього псевдоклас *:hover*:

```
#navbar ul { display: none; }
#navbar li:hover ul { display: block; }
```

Приберемо у обох списків відступи і маркери, встановлені за замовчуванням. Елементи списку з навігаційними посиланнями робимо плаваючими, формуючи горизонтальне меню, але для елементів списку, що містять підпункти, задаємо *float: none;*, щоб вони відображалися один під одним:

```
#navbar, #navbar ul {
        margin: 0;
        padding: 0;
        ist-style-type: none;
}
#navbar li { float: left; }
#navbar ul li { float: none; }    /* Щоб пункти меню відображалися один під одним */
```

Тепер необхідно зробити так, щоб підміню, яке випадає, не зміщало вниз контент, розташований під панеллю навігації. Для цього задамо для пунктів списку властивість *position: relative;*, а списку, який містить підпункти, – *position: absolute;* . Так само додаймо властивість *top* зі значенням 100%, щоб абсолютно позиціоноване підменю відображалося точно під посиланням:

```
#navbar ul {
	display: none;
	position: absolute;
	top: 100%;
}
#navbar li {
	float: left;
	position: relative;
}
#navbar { height: 30px; }
```

Висота для батьківського списку додана спеціально. Так як браузери не враховують як вміст елементу плаваючий контент, то без додавання висоти список буде проігнорований браузером і контент, який слідує за списком, буде обтікати меню.

Тепер залишилося стилізувати обидва списки і меню, що випадає, буде готово:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Название документа</title>
		<style>
			#navbar ul{
				display: none;
				background-color: #f90;
				position: absolute;
				op: 100%;
			}
			#navbar li:hover ul { display: block; }
			#navbar, #navbar ul{
				margin: 0;
				padding: 0;
				list-style-type: none;
			}

			#navbar {
				height: 30px;
				background-color: #666;
```

```
                padding-left: 25px;
                min-width: 470px;
            }

            #navbar li {
                float: left;
                position: relative;
                height: 100%;
            }
            #navbar li a {
                display: block;
                padding: 6px;
                width: 100px;
                color: #fff;
                text-decoration: none;
                text-align: center;
            }
            #navbar ul li { float: none; }
            #navbar li:hover { background-color: #f90; }
            #navbar ul li:hover { background-color: #666; }
        </style>
    </head>
</html>
```

2.4 Виконання завдання

2.4.1 Практичне заняття виконується в рамках веб-сайту, визначеного в якості індивідуального завдання при виконанні практичного заняття № 1.

2.4.2 Додайте на головної сторінці головний навігаційний блок сайту (меню). Хоча б один пункт цього меню повинен бути випадаючим меню. Зробить з кількох пунктів меню працюючі посилання (в тому числі на другому рівні меню). Створить для цього відповідно до зазначеної функціональності Вашого сайту кілька додаткових сторінок (вони можуть містити тільки назву сайту, назву сторінки та логотип сайту).

2.4.3 Зробіть назву та логотип сайту, назву сторінки і меню нерухливим, тобто що залишається на місці при прокручуванні.

2.4.4 На одну з доданих (нових) сторінок додайте назву та логотип сайту, назву сторінки, головне меню та кілька інформаційних блоків з контентом.

Пункт меню, який відповідає поточної сторінці, повинен бути виділений якимось чином і не повинен бути гіперпосиланням.

2.4.5 Логотип сайту на доданої сторінці зробіть гіперпосиланням на головну сторінку сайту.

2.4.6 Складіть звіт, оформіть його відповідно до ДСТУ 3008: 2015 [4] і здайте роботу викладачеві.

2.4.7 Звіт повинен містити:

– номер індивідуального завдання та найменування сайту;

– HTML-код та CSS-код всіх розроблених і змінених сторінок в межах цього практичного завдання;

– скріншоти вікон браузера з розробленими і оновленими веб-сторінками, які демонструють виконання завдань до практичного заняття.

2.5 Контрольні запитання

1. Як формується нормальний потік?
2. У якому випадку використовують відносне позиціювання елементу?
3. Який вид позиціонування дозволяє зробити елемент сторінки нерухомим?
4. Як створюють посилання на елемент поточної сторінки?
5. Які можливості надає властивість *overflow* при переповненні блоку?
6. Як можна зображення створити гіперпосиланням?
7. Для чого служить тег <*area*>?
8. Яку форму може мати активна область зображення?
9. Які засоби CSS дозволяють додавати генерований уміст в документ?
10. Для чого використовують псевдокласи в CSS?
11. Відносно чого будуються відносні посилання?
12. Як будується селектор нащадків (контекстний)?
13. Яка роль ключового слова *inherit* при установці його властивості CSS?
14. Яка роль індикатора *!Important*?
15. До яких елементів сторінки застосовує стилі псевдоклас *:visited*?
16. У який момент відображаються стилі елемента за допомогою псевдокласу *:hover**?***
17. Яке призначення структурних псевдокласів?
18. Як можна використовувати псевдоелементи *::before* і *::after*?

3 СТВОРЕННЯ АДАПТИВНИХ ВЕБ-СТОРІНОК ЗАСОБАМИ CSS

3.1 Мета заняття

Мета заняття полягає в ознайомленні з можливостями і набутті практичних навичок роботи з модулем CSS Flexbox, модулем CSS Grid, медіазапитами для створення адаптивних веб-сторінок.

3.2 Методичні вказівки з організації самостійної роботи студентів

Готуючись до заняття, необхідно повторити лекційний матеріал, який висвітлює методи верстки адаптивних web-сторінок, можливості та особливості використання модуля макета гнучкого контейнера Flexbox, контейнера Grid та медіазапитів.

3.3 Теоретичні положення

3.3.1 Тонкощі використання коефіцієнтів гнучкості модуля CSS Flexbox

Коефіцієнти гнучкості задають за допомогою властивостей *flex-grow*, *flex-shrink*, *flex-basis* і скороченої властивості *flex*, яка поєднує зазначені вище.

Використання властивості у вигляді *flex: 1*; дозволяє *flex*-елементу розтягуватися, займаючи обсяг пам'яті, який доступний.

CSS-властивість *flex-grow* використовується для настройки так званого «коефіцієнта зростання» елементів, що дозволяє цим елементам розтягуватися, займаючи обсяг пам'яті, який доступний. Як значення цієї властивості можна використовувати тільки цілі числа.

Властивість *flex-grow* може впливати на ширину або на висоту елемента, в залежності від значення властивості *flex-direction*.

Без використання *flex-grow* ширина (висота) *flex*-елементів буде рівна їх вихідної ширині або висоті (значення за замовчуванням).

Значення *flex-grow: 1;* означає, що вільний простір, який є, розподіляється між елементами.

Ширина елемента *flex*-елементів обчислюється за формулою:

(1)

$$\left(\left(\frac{flex-grow \text{ елемента}}{\text{сума значень } flex-grow}\right) * \text{доступний простір}\right) + \text{вихідна ширина елемента}$$

Значення *flex-grow: 0;* означає, що *flex*-елемент не змінюватиме розміри, займаючи певну частину вільного простору контейнера. Цей прийом може бути корисний в тих випадках, коли потрібно, щоб якийсь *flex*-елемент зберігав би свою вихідну ширину.

Сенс використання *flex-grow* полягає в розподілі доступного простору між елементами.

Властивість *flex-shrink* дозволяє задати так званий «коефіцієнт стиснення» елемента. Якщо розмір всіх flex-елементів більше, ніж розмір їх контейнера, розмір елементів буде зменшений відповідно до призначених ним значенням властивості *flex-shrink*.

Встановимо для одного з flex-елементів такі стилі:

```
.item-2 {
        width: 300px;
        flex-shrink: 1;
}
```

В цьому випадку браузер зробить ширину елемента item-2 рівним 300px при виконанні наступних умов:

– загальна ширина всіх елементів менше ширини контейнера;

– ширина області перегляду сторінки дорівнює або більше ширини елемента.

Елемент з прикладу вище не буде стискатися, зберігаючи ширину в 300px до тих пір, поки йому вистачає місця.

3.3.2 Встановлення розмірів *flex*-елементу

Властивість *flex-basis* задає базовий розмір *flex*-елемента, який він має до розподілу вільного простору відповідно до коефіцієнтів гнучкості, що задаються властивостями *flex-grow* і *flex-shrink*. За замовчуванням, для всіх значень, крім *auto* і *content*, значення *flex-basis* дорівнює ширині елемента, а при використанні властивості *flex direction: column;* – його висоті.

Властивість *flex-basis* приймає ті ж значення, які можуть приймати властивості *width* і *height*. Значенням, що задається за замовчуванням, є *auto*, яке встановлюється на основі значення content. Значення content задається автоматично, на основі розміру вмісту *flex*-елемента.

Розглянемо наступний приклад:

```
.item-1 {
        flex-grow: 0;
        flex-shrink: 0;
        flex-basis: 50%;
}
```

Для того, щоб розмір елемента не перевищив би 50%, важливо скинути в 0 властивість *flex-grow*.

Якщо замість цього властивісті *flex-basis* встановити значення 100%, то елемент займе 100% ширини батьківського елементу, а інші елементи будуть перенесені на новий рядок. Це дійсно в разі, коли для інших *flex*-елементів властивість *flex-basis* не встановлено.

3.3.3 Властивість *flex*

Властивість *flex* дозволяє в скороченому форматі задавати значення властивостей *flex-grow*, *flex-shrink* і *flex-basis*. Значення за замовчуванням, яке приймає це властивість, є *auto*. Воно відповідає *flex: 0 1 auto*. Це означає, що *flex*-елементи будуть збільшуватися в розмірах при збільшенні розмірів їх умісту.

3.4 Виконання завдання

3.4.1 Практичне заняття виконується в рамках веб-сайту, визначеного в якості індивідуального завдання при виконанні практичного заняття № 1.

3.4.2 Необхідно виконати адаптацію двох з вже створених сторінок сайту до зміни ширини вікна та до зміни розміру шрифту таким чином, що б вигляд сторінки істотно не змінювався. Для цього потрібно використати медіазапити, засоби модуля CSS Flexbox або модуля CSS Grid.

3.4.3 При зменшенні вікна браузера необхідно в переломних точках дизайну зменшувати розміри зображень (можливо лише деяких), розмір шрифту

текстів, розміри деяких блоків, зменшувати вільний простір меж елементами, щоб вміст сторінки вміщувався в межах області перегляду.

3.4.4 При збільшенні вікна браузера необхідно збільшувати розміри зображень, розмір шрифту текстів, розміри деяких блоків, зменшувати вільний простір між елементами, щоб розміщення елементів не було дуже розрідженим, а рядки тексту дуже довгими.

3.4.5 Продемонструйте результати роботи викладачеві в браузері.

3.4.6 Складіть звіт, оформіть його відповідно до ДСТУ 3008: 2015 [4] і здайте роботу викладачеві.

3.4.7 Звіт повинен містити:

– номер індівідуального завдання та найменування сайту;

– HTML-код та CSS-код всіх змінених сторінок в межах цього практичного завдання;

– скріншоти вікон браузера з розробленими і оновленими веб-сторінками, які демонструють виконання завдань практичного заняття.

3.5 Контрольні запитання

1. У чому полягає перевага модуля Flexbox при створенні адаптивних сторінок в порівнянні з підходом на основі обтікання і позиціонування?
2. Яке призначення властивості justify-content (модуль Flexbox)? Які умови його використання?
3. Які можливості щодо адаптації сторінки до ширини вікна надає модуль Grid?
4. Чим можуть бути представлені умови медіа-запиту?
5. До яких частин коду веб-сторінки можна додавати медізапити?
6. Які можливості щодо адаптації сторінки надають медіазапити?
7. Як визначити стиль елементу в залежності від ширини області перегляду?
8. Для чого використовують псевдокласи в CSS?
9. Чи можна управляти накладенням елементів один на одного при позиціонуванні?
10. Як можна перешкоджати зайвому розростанню або зменшенню рідкого макета?
11. Які можливості модуля макета гнучкого контейнера Flexbox?
12. Чому відповідає одиниця виміру *em* в CSS?

13. Чому відповідає одиниця виміру *vm* в CSS?

14. У чому полягає сенс використання властивості *flex-grow*?

15. З чого складаються значення властивостей блоку *width* і *height*?

16. При якому вигляді позиціонування елемент зміщується щодо свого початкового положення?

4 СТВОРЕННЯ ФОРМ ЗАСОБАМИ HTML ТА ОБРОБКА ПОДІЙ ЗАСОБАМИ JAVASCRIPT

4.1 Мета заняття

Мета заняття полягає в ознайомленні з елементами управління HTML, механізмами обробки подій, способів динамічної зміни властивостей CSS елементів сторінки, особливостями роботи з функціями JavaScript і набутті практичних навичок роботи з ними.

4.2 Методичні вказівки з організації самостійної роботи студентів

Готуючись до практичного заняття, необхідно повторити лекційний матеріал, який висвітлює особливості створення форм, синтаксиса и типов данных мови JavaScript, обробки подій і динамічної зміни значень атрібутів єлементів сторінки.

4.3 Теоретичні положення

4.3.1 Особливості елемента керування <input>

HTML тег <label> визначає текстову мітку для елемента <input>. За своїм виглядом мітка являє собою звичайний текст.

Якщо елемент <input> помістити всередину елемента <label>, то користувач зможе вибрати дане поле введення (встановити курсор) кліком по тексту мітки, а не по самому полю введення.

У разі, коли елемент <input> не всередині елемента <label>, атрибут for тега <label> дозволяє пов'язати мітку з полем введення. Значення атрибута for повинно бути еквівалентно значенням атрибута id того елемента форми, до якого буде ставитися мітка. При встановленні такого зв'язку користувач зможе вибрати поле введення кліком по тексту мітки.

Атрибут *list* елемента <input> вказує на список варіантів, створений за допомогою тега <datalist>, які можна вибирати при наборі тексту. Спочатку цей список прихований і стає доступним при отриманні полем фокусу.

Атрибути *max* і *min* елемента <input> встановлюють верхнє й нижнє значення для введення числа (для *type = "number"*, t*ype = "range")* або дати (для *type = "date"*) в поле форми. Синтаксис для встановлення верхніх значень:

```
<input type="number" max="максимальное число">
<input type="range" max="максимальное число">
<input type="date" max="верхняя дата">
```

Для визначення нижніх значень використовується аналогічній синтаксис.

Дата в атрибути *max* і *min* та в атрибут *value* записується в форматі РРРР-ММ-ДД (наприклад: 2012-12-22).

Атрибут *pattern* елемента <input> дозволяє вказати шаблон регулярного виразу, згідно з яким потрібно вводити і перевіряти дані в поле форми. Шаблон регулярного виразу повинен відповідати введеному значенню цілком. Якщо присутній атрибут pattern, то форма не буде відправлятися, поки не буде заповнене правильно.

Для створення регулярного виразу використовується спеціальна мова регулярних виразів.

Атрибут *pattern* можна використовувати для елемента <input> з типом: *text, search, url, tel, email* и *password.*

Приклад:

```
<p><input type="tel" pattern="2-[0-9]{3}-[0-9]{3}"></p>
```

Атрибут *value* елемента <input> вказує значення елементу <input> і використовується по-різному для різних типів введення:

- для *"button"*, *"reset"*, і *"submit"* – визначає текст на кнопці;

- для *"text", "password"*, *"hidden"*, *"date"* – визначає початкове (за замовчуванням) значення поля введення;

- для *"checkbox"*, *"radio"*, *"image"* – визначає значення, пов'язане з введенням (значення, яке передається серверу при відправці).

Атрибут *value* не може використовуватися з *type = "file"*, і обов'язково повинен бути присутнім при *type = "checkbox"* і *type = "radio"*.

Атрибут *placeholder* елемента <input> вказує текст для виведення всередині поля форми, який зникає при отриманні фокусу.

Атрибут *required* елемента <input> встановлює поле форми обов'язковим для заповнення перед відправкою форми на сервер. Якщо обов'язкове поле пусте,

браузер виведе повідомлення, а форма відправлена не буде. Вид і зміст повідомлення залежить від браузера і змінюватися не може.

Атрибут *autocomplete* включає/відключає автозаповнення поля. Цей атрибут був доданий в HTML5.

Якщо автозаповнення включено, то при введенні перших букв тексту виводиться список значень, які вводилися в це поле раніше, і з якого можна вибрати необхідне.

Синтаксис визначення атрибута:

```
<input autocomplete="on | off">
```

За замовчуванням автозаповнення поля включено.

Атрибут *autocomplete* працює з наступними типами елемента <input>: *text*, *search*, *url*, *tel*, *email*, *password*, *datepickers*, *range*, *color*.

4.3.2 Особливості валідації форм

Надсилання даних серверу відбувається при натисканні користувачем кнопки типу "*submit*" або клавіші "*Enter*", коли фокус знаходиться в полі введення. При цьому ініціюється ряд подій, але валідацію форми (задану перевірку правильності введення) браузер виконує за подією форми "*submit*" у дії за умовчанням (якщо вона не була скасована одним із обробників події). Таким чином, браузер виконує валідацію форми вже після виконання будь-якого зареєстрованого обробника події "*submit*".

Якщо результати валідації форми браузером повинні бути відомі раніше, для того щоб можна було виконати будь-які дії, що залежать від результатів валідації, можна використовувати метод елемента (в даному випадку форми) *checkValidity()*.

Метод *checkValidity()* елемента форми виконує задану в елементах введення (в коді HTML) валідацію елементів форми та повертає *true*, якщо всі елементи форми пройшли валідацію, інакше повертає *false*. Якщо форма не валідна, цей метод також запускає на ній подію *invalid*.

У разі використання методу *checkValidity()* необхідно скасувати стандартну обробку події "submit" за допомогою методу *preventDefault()* об'єкта події *Event*, щоб уникнути повторної валідації форми браузером.

Необхідно пам'ятати, що при скасуванні стандартної обробки події "*submit*" скасовується і надсилання даних серверу. У таких випадках запит на надсилання даних формується "вручну" з використанням можливостей API асинхронного обміну даними (AJAX).

4.3.3 Особливості блоку

Блок типу не має відступів від інших елементів, тобто елементи, розташовані до і після нього, будуть впритул прилягати зверху і знизу. Відступи легко можна відрегулювати за допомогою каскадних таблиць стилів (CSS).

4.3.4 Глобальний об'єкт *Window*

Глобальний об'єкт надає змінні і функції, доступні в будь-якому місці програми. За замовчуванням це ті об'єкти, які вбудовані в мову або середовище виконання.

Якщо середовище виконання браузер, то глобальний об'єкт зветься Window. Для звернення до цього об'єкта визначена наперед глобальна змінна window.

До всіх властивостів глобального об'єкта можна звертатися безпосередньо, наприклад:

```
var gVar = 5;

alert(window.gVar); // 5 (становится свойством глобального
объекта)
```

4.3.5 Опис подій форми

Події, які найчастіше застосовуються до елементів форми, наведені в табл.4.1.

Таблиця 4.1 – Перелік і опис подій, що застосовуються до елементів форми

Подія	Описание
blur	Скрипт запуститься, коли елемент втратить фокус
change	Скрипт запуститься після закінчення зміни значення елемента форми, коли ця зміна зафіксована (при втраті фокусу або відразу при виборі значення)
contextmenu	Скрипт запуститься при виклику контекстного меню
focus	Скрипт запуститься, коли елемент отримає фокус
formchange	Скрипт запуститься, коли змінюється будь-яке з полів форми. Може бути вказано як у тега form так і у дочірніх елементів (input, textarea і т.д.)
forminput	Скрипт запуститься, коли користувач буде вводити дані в форму. Викликається відразу ж після зміни, не чекаючи, коли поле втратить фокус. Діє відразу для всієї форми
input	Скрипт запуститься відразу при зміні значення текстового елемента
invalid	Скрипт запуститься, коли елемент, що відправляється, був перевірений, але його вміст не задовольнив встановленим обмеженням
select	Скрипт запуститься якщо у текстовому полі форми було виділено текст за допомогою миші або клавіатури
submit	Скрипт запуститься при відправці форми (після натискання кнопки типу "submit" або клавіші "Enter", коли фокус знаходиться в полі введення)

4.4 Виконання завдання

4.4.1 На одній зі сторінок сайту відповідно до функціональності сайту створіть форму (після погодження її з викладачем), яка містить наступні елементи:

- поля введення даних;

- пояснюючі написи до полів вводу;

- кнопку для підтвердження завершення введення даних;

- кнопку очищення всіх полів з даними.

4.4.2 Поля, які є обов'язковими для введення, необхідно якимось чином позначити. Якщо таких полів більше, ніж необов'язкових, то помітити краще необов'язкові поля.

4.4.3 Для тих полів, де можливо, забезпечте вибір значень зі списку або інші способи вибору значень замість ручного введення.

4.4.4 Для події натискання кнопки підтвердження завершення введення необхідно реалізувати перевірку на заповненість обов'язкових полів та перевірку на правильність введених даних (по можливості) засобами HTML та браузера.

4.4.5 Для події натискання кнопки очищення полів необхідно реалізувати очистку від введених даних усіх полів форми.

4.4.6 За подією надсилання даних форми при відсутності помилок введення даних, необхідно відобразити спливаючий блок з текстом, що повідомляє про те, що введені дані прийняті в обробку. Спливаючий блок повинен мати символ закриття (x), при кліці по якому він має зникнути.

4.4.7 Реєстрацію обробників подій необхідно виконати лише за допомогою передачі обробників методу addEventListener() об'єкта.

4.4.8 Продемонструйте результати роботи викладача в браузері.

4.4.9 Складіть звіт, оформіть його відповідно до ДСТУ 3008: 2015 [4] та захистите роботу.

4.4.10 Звіт повинен містити:

– найменування інформаційної системи;

– HTML-код, CSS-код і JS-код сторінки сайту, що містить форму;

– скріншоти вікна браузера, що відображають вид сторінки до введення даних і з реакцією програми на помилки або відсутність даних у обов'язкових полях, на відправку правильних даних.

4.5 Контрольні запитання

1. Чим відрізняється метод передачі даних форми POST від методу GET?

2. Для чого служить атрибут форми action?

3. У якому вигляді передаються дані форми на сервер при використанні методу передачі GET?

4. Які недоліки має метод передачі даних форми GET?

5. Для чого служить атрибут tabindex елементів управління?

6. Яке призначення елементів <fieldset> і <legend>?

7. В який момент відбувається предача даних форми на сервер?

8. Що означає спливання події?

9. Який порядок управління для неспливаючі події?

10. Чи може бути скасовано дію за замовчуванням при виникненні події?

11. Який спосіб реєстрації оброблювача події дозволяє зареєструвати кілька оброблювачів події?

12. Де міститься інформація про подію, яка виникла, для обробника події?

13. Яким чином можна отримати доступ до об'єкта події Event?

14. Які способи визначення функції існують в JavaScript?

15. Які особливості визначення параметрів функції і зазначення аргументів при виклику функції?

16. Які функції називають функціями зворотного виклику?

17. Чим виконання функцій зворотного виклику відрізняється від виконання звичайних функцій?

18. Як пов'язані змінні і функції, які оголошені глобально, з об'єктом Window?

5 ЗМІНА WEB-СТОРІНОК З ВИКОРИСТАННЯМ ЗАСОБІВ DOM

5.1 Мета заняття

Мета практичного заняття полягає у вивченні можливостей об'єктів API для HTML- и XML-документів DOM і набутті практичних навичок з динамічної зміни веб-документів засобами DOM.

5.2 Методичні вказівки з організації самостійної роботи студентів

Готуючись до практичного заняття, необхідно повторити лекційний матеріал, який висвітлює призначення та можливості API DOM щодо зміни складу елементів сторінки та CSS-властивостів елементів.

5.3 Теоретичні положення

5.3.1 Використання *data*-атрибутів

data-атрибути – це атрибути, які можна застосовувати для зручного зберігання різної корисної програмісту інформації в стандартних HTML-елементах. Така інформація, зокрема, доступна і може застосовуватися в JavaScript- і CSS-кодах.

Мова йде про створення власних атрибутів, назви яких починаються з префікса data-. Частина імені атрибута, яка йде після префікса, може називатися так, як потрібно розробнику.

Приклад створення data-атрибутів:

```
<article
  id="electriccars"
  data-columns="3"
  data-index-number="12314"
  data-parent="cars">
...
</article>
```

Для читання data-атрибутів в JavaScript можна використовувати метод getAttribute() об'єкта елемента сторінки з аргументом, рівним повному імені атрибута.

Інший спосіб звернення до data-атрибутів – використання об'єкта `dataset`, що є подоб'екти об'єкта елемента сторінки. У цьому випадку необхідно звернутися до властивості об'єкта `dataset` з ім'ям, рівним частини імені атрибута після префікса data- (дефіси в імені властивості видаляються, а перша буква слова після дефіса в імені перетвориться в верхній регістр (camelCase).

Приклад:

```
var article = document.getElementById('electriccars');

article.dataset.columns // "3"
article.dataset.indexNumber // "12314"
article.dataset.parent // "cars
```

В CSS доступ до data-атрибутів здійснюється так само, як і до звичайних HTML-атрибутів. Їх можна використовувати, наприклад, в селекторах атрибутів в CSS для зміни стилів у відповідності зі значенням атрибута.

Приклад:

```
article[data-columns='3']{
  width: 400px;
}
article[data-columns='4']{
  width: 600px;
}
```

Крім того, data-атрибути можна використовувати у властивості content разом з псевдоелементи `::before` і `::after` для вставки значень таких атрибутів, наприклад:

```
article::before {
  content: attr(data-parent);
```

5.3.2 Методи відкладеного виконання функцій

Для виконання дій через певні проміжки часу (планування виклику) в об'єкті *Window* передбачені методи відкладеного виконання функцій.

Метод *setTimeout (func | code, [delay], [arg1], [arg2], ...)* дозволяє викликати функцію один раз через певний інтервал часу, тобто дозволяє зробити затримку перед запуском коду. Він має такі параметри:

func|code – ім'я функції (без лапок і круглих дужок) або визначення функції, або рядок коду для виконання;

delay – певну затримку перед запуском в мілісекундах (1000 мілісекунд = 1 секунда). При значенні 0 планувальник буде викликати функцію тільки після завершення виконання поточного коду;

arg1, arg2… – аргументи, що передаються в функцію.

Метод повертає ідентифікатор таймера, який може знадобитися для скасування таймера.

Скасувати таймер можна методом *clearTimeout(timerId).*

Метод *setInterval (func|code, [delay], [arg1], [arg2], ...)* дозволяє запускати на виконання зазначену функцію періодично через заданий інтервал часу. Повертає він ідентифікатор таймера.

Щоб зупинити подальше періодичне виконання функції, необхідно викликати метод *clearInterval(timerId)*.

Приклад:

```
// повторить с интервалом 2 секунды
let timerId = setInterval(() => alert('tick'), 2000);

// остановить вывод через 5 секунд
setTimeout(() => { clearInterval(timerId);
alert('stop'); }, 5000); }
```

Інший спосіб регулярного запуску функції: використання рекурсивного виклику *setTimeout*. З його допомогою наступний виклик може бути заданий по-різному в залежності від результатів попереднього.

Зупинити такий таймер можна просто не давши запуститися методу *setTimeout*, наприклад, через *if*.

Приклад:

```
<input type="text" value="1" id="test">
<input type="submit" value="start" onclick="timer()">

function timer() {
   //получаем инпут
```

```
    var elem = document.getElementById('test');
    //увеличиваем значение атрибута на единицу
    elem.value = parseInt(elem.value)+1;

    //если в инпуте меньше 10-ти - то работаем дальше
    if(elem.value < 10) {
        //Функция timer запускает саму себя с задержкой в 1 секунду
        window.setTimeout(timer, 1000);
    }
}
```

У прикладі вище значення атрибута `value` елемента `<input>` кожну секунду збільшується на одиницю. Натискання на кнопку `start` запускає функцію `timer`, яка збільшує значення `input` на 1 і після цього викликає саму себе з затримкою в 1 секунду. Так триває до тих пір, поки значення `value` не стане рівне десяти.

5.3.3 Використання слайдерів на сайтах

Слайдер – це блок певних розмірів і з певним (вигідним для сприйняття) розміщенням на сторінці, який містить контент, який змінюється. Це можуть бути зображення, відео, текстові блоки, посилання. Слайдер може працювати як в ручному, так і автоматичному режимі, змінюючи слайди через певний часовий інтервал.

Слайдери є візуально привабливими для відвідувачів веб-сайту і стимулюють інтерес цільових користувачів до різних матеріалів, розміщених на сайті. Крім того, використання слайдерів дозволяє заощадити місце, так як один блок слайдера, дозволяє продемонструвати кілька матеріалів сайту.

Слайдери для сайтів можуть мати різний функціонал. Вони можуть працювати в режимі послідовного перегляду зі зміною слайдів через деякий проміжок часу, можуть давати можливість для «перегортання» сторінки вручну, можуть забезпечувати можливість переходу на матеріал або показ у повний розмір і ін. Деякі слайдери реалізують паузу при наведенні курсору на слайд.

Зазвичай для слайдерів використовують 3-5 слайдів. Як правило, слайдери включають такі елементи:

- екран для слайдів (елементів контенту);

- засоби навігації, що дозволяють перейти до чергового або попереднього слайду;

- маркери, що відображають загальну кількість слайдів і положення поточного слайда в ланцюжку слайдів.

Деякі слайдери можуть включати додаткові елементи:

- мініатюри невидимих в поточний момент слайдів;

- таймер, що відображає час, який залишився до зміни слайда.

Розміщують слайдери, як правило, на головній сторінці.

На рис. 5.1 наведено приклад слайдера на сайті.

Рисунок 5.1 – Приклад слайдера на сайті

Послідовність слайдів необхідно добре продумувати. Зазвичай першим слайдам користувач приділяє набагато більше уваги, ніж наступним. Тому на першому слайді потрібно розміщувати найважливіший контент. Слайди повинні бути розставлені за ступенем важливості.

Слайдер не повинен бути єдиною точкою доступу до контента і функцій, розташованим на ній. Будь-яку важливу інформацію з слайдера варто продублювати десь ще на сторінці, щоб підвищити шанси, що її побачать.

5.4 Виконання завдання

5.4.1 ПЗ виконується в рамках веб-сайту, визначеного в якості індивідуального завдання в ПЗ № 1.

5.4.2 Отримайте індивідуальне завдання у викладача.

5.4.3 Розробіть HTML-код, CSS-код і JS-код, необхідний для виконання індивідуального завдання.

5.4.4 Продемонструйте результати роботи викладачеві в браузері.

5.4.5 Складіть звіт, оформіть його відповідно до ДСТУ 3008: 2015 [4] і здайте роботу викладачеві.

5.4.6 Звіт повинен містити:

– номер індивідуального завдання і найменування сайту;

– HTML-код, CSS-код і JS-код всіх розроблених для цього ПЗ і змінених сторінок;

– скріншоти вікон браузера, що демонструють результати виконання завданнь до ПЗ.

5.5 Контрольні завдання

Контрольні завдання діляться на кілька категорій, кожна з яких має спільні риси, незалежно від найменування сайту.

Опис індивідуальних завдань до цього ПЗ потрібно переглядати в окремому файлу.

5.5.1 Вимоги до реалізації корзини з вибраними для покупки товарами

При виконанні такого типу завдання необхідно виконати такі умови:

1. На сторінці, що відображає конкретні товари для покупки, необхідно відобразити найменування товару, його вартість і кнопку (або гіперпосилання без атрибута *href*) "*Заказать*".

2. При натисканні на *"Замовити"* необхідно реалізувати відкриття спливаючого блоку *"Корзина"*. Спливаючій блок повинен містити найменування товару, його вартість і кнопки (гіперпосилання) *"Оформити замовлення"*,

"Продовжити покупки". Рядок з товаром в корзині повинен мати символ хрестика для видалення його з корзини.

3. При додаванні в корзину ще одного товару в ній повинні зберігатися і раніше додані товари.

5. При натисканні на "*Продовжити покупки*" вікно корзини має зникати.

5.5.2 Вимоги до реалізації слайдера

При виконанні цього типу завдання необхідно виконати такі умови:

1. Необхідно реалізувати тип слайдера, заданий в індивідуальному завданні.

2. Слайдер обов'язково повинен включати маркери, що відображають загальну кількість слайдів і положення поточного слайда в ланцюжку слайдів.

5.5.3 Вимоги до реалізації тестування або опитування

Тест або опитування може містити будь-яку кількість питань, більше ніж два.

Необхідно відображати на одній веб-сторінці тільки одне питання, кілька радіокнопок або чекбоксів з варіантами відповіді і кнопку "Продовжити".

Після натискання кнопки "Продовжити" мусить відобразитися чергове запитання.

При відображенні останнього питання кнопка повинна змінити напис на "Показати результати", при натисканні на яку необхідно відобразити наступну інформацію:

- для тестування – кількість правильних відповідей, кількість невірних відповідей та інформацію про те, пройдений тест чи ні;
- для опитування – узагальнені результати опитувань із зазначенням, наприклад, кількісних показників прихильників чогось, противників і хто не визначився.

Все тестування або опитування повинно бути реалізовано на одній сторінці, вміст якої має динамічно змінюватися.

5.6 Контрольні запитання

1. На основі чого утворюються текстові вузли в дереві вузлів HTML-документа?
2. Як можна отримати доступ до об'єктів *Document* і *Window*?
3. Які існують способи отримання об'єкта елемента веб-сторінки?
4. Чим відрізняється дія методу *getElementsByTagName()* об'єктів *Document* і *Element*?
5. Що повертає метод *getElementsByClassName()* об'єкта *Document*?
6. Яким чином утворюються імена DOM-властивостей, які відповідають атрибутам елементів сторінки?
7. Як називається DOM-властивість, які відповідає атрибуту *class*?
8. Як відрізняються значення атрибутів елементів від значень відповідних DOM-властивостей по типу даних?
9. Які існують можливості при роботі безпосередньо з атрибутами елементів сторінки засобами DOM?
10. Як можна використовувати властивість innerHTML об'єкта Element?
11. Чим відрізняється властивість outerHTML об'єкта Element від властивості innerHTML?
12. Що необхідно зробити після створення об'єкта вузла елемента веб-документа, щоб елемент став частиною документа і відобразився на сторінці?
13. Для чого служить метод removeChild () об'єкта Node і щодо якого об'єкта він викликається?
14. Для чого служить DOM-властивість *disabled* елементів <style> і <link>?
15. Що представляє об'єкт *CSSStyleSheet*?
16. Що містить властивість *document.styleSheets*?
17. Для чого використовують атрибут *title* елементів <style> і <link>?
18. Що містить властивість *cssRules []* об'єкта *CSSStyleSheet*?
19. Що містить властивість *selectorText* об'єкта *CSSRule*?
20. Що представляє об'єкт *CSSStyleDeclaration***?**
21. 21. Що містить властивість *cssText* об'єкта *CSSStyleDeclaration*?
22. Яке призначення методу *insertRule ()* об'єкта *CSSStyleSheet*?
23. Як засобами DOM створити нову таблицю стилів в веб-документі?
24. Що таке *data*-атрибути і як можна їх використовувати?
25. Для чого служить метод *setInterval ()* об'єкта *Window*?

6 РОБОТА З ЛОКАЛЬНИМ СХОВИЩЕМ ДАНИХ ЗАСОБАМИ JAVASCRIPT

6.1 Мета заняття

Мета практичного заняття полягає у вивченні можливостей і придбанні практичних навичок по роботі з локальними сховищами даних Web Storage.

6.2 Методичні вказівки з організації самостійної роботи студентів

Готуючись до практичного заняття, необхідно повторити лекційний матеріал, який висвітлює можливості та засоби для роботи з локальним сховищем даних Web Storage.

6.3 Теоретичні положення

6.3.1 Особенності використання веб-сховищ

Зберігати в веб-сховище (Web Storage) можна все те, що може бути перетворено в строкові дані.

У веб-сховище зазвичай зберігають такі дані:

- стан роботи програми – наприклад, останній відкритий документ, значення зміненого поля введення або історію пошуку по сайту;
- настройки користувача – наприклад, обрана тема оформлення сайту або вид відображення інформації;
- збереження даних, які ввів користувач при невдалій спробі відправки на сервер.

Конфіденційну інформацію користувача не слід зберігати в веб-сховище, так як дані не шифруються і можуть бути скомпрометовані.

Веб-сховище надає набагато більший розмір пам'яті, ніж cookie. Розмір веб-сховища залежить від типу браузера і варіюється від 2500 Кб до 5000 Кб.

Отримати всі дані, які записані в сховище, можна за допомогою циклу:

```
for (var i = 0; i < localStorage.length; i++) {
  var key = localStorage.key(i);
  console.log(key + ' = ' + localStorage[key]);
}
```

У циклі перебираються всі елементи сховища, читаються їх ключі і значення.

Якщо в веб-сховище необхідно зберігати не тільки строкові дані, то доведеться (при читанні даних обов'язково) виконати перетворення типів.

При зберіганні дати необходімо використовувати об'єкт *Date*.

Приклад запису дати в сховище і читання її зі сховища :

```
// Запис в сховище з форматуванням дати в текстовий
// рядок
var currentTime = new Date();
localStorage["date"] = currentTime.getFullYear() + '/' +
currentTime.getMonth()+ '/' + currentTime.getDate();

// Читання зі сховища
var data = new Date(localStorage["date"]);
console.log(data.getDate() + ' ' + data.getMonth() + ' '
+ data.getFullYear());
```

Об'єкт типу *Date*, створений за допомогою конструктора, описує поточну дату.

Для запису масиву в сховище необхідно виконати наступні дії:

- перетворення масиву в рядок;
- запис рядка в сховище;
- читання рядка зі сховища;
- перетворення рядка в масив.

Для перетворення масиву в рядок можна використовувати метод JSON.stringify, при читанні масиву зі сховища використовують метод JSON.parse, наприклад:

```
var data = JSON.parse(localStorage["myArray"]);
```

При зберіганні об'єктів в веб-сховище використовують ті ж методи об'єкта *JSON*, що і при зберіганні масиву.

Перед використанням веб-сховища необхідно перевірити, що воно підтримується і доступно в поточному браузері.

Браузери, що підтримують *Web Storage* матимуть властивість *localStorage* об'єкта *Window*. Тому для перевірки того, що веб-сховище підтримується браузером, досить виконати перевірку на наявність властивості *localStorage*.

Проте, якщо, наприклад, localStorage існує, це ще не дає гарантії, що воно доступно, тому що різні браузери мають настройки, які дозволяють відключити його. Тому браузер може підтримувати localStorage, але при цьому веб-сховище може бути недоступним для скриптів на сторінці.

Надійний спосіб перевірити, що сховище доступно – це зробити тестову запис в сховище з обробкою винятку.

Приклад виконання перевірки доступності веб-сховища:

```
function storageAvailable(type) {
 try {
       var storage = window[type];
             x = '__storage_test__';
       storage.setItem(x, x);
       storage.removeItem(x);
       return true;
 }
 catch(e) {
       return false;
 }
}

if (storageAvailable('localStorage')) {
 // Код обращения к хранилищу
}
else {
 // Код сообщения о том, что локальное хранилище не
доступно
}
```

6.3.2 Робота з прокруткою

6.3.2.1 Отримання поточної прокрутки

У звичайного елемента поточну прокрутку можна отримати у властивостях *scrollLeft* (піксель по горизонталі) і *scrollTop* (піксель по вертикалі) об'єкта елемента.

Для отримання прокрутки сторінки більшість браузерів коректно обробить запит до властивостей *documentElement.scrollLeft / scrollTop*, проте в *Safari / Chrome / Opera* є помилки, через які слід використовувати *document.body.*

Прокрутку сторінки можна також отримати за допомогою властивостей *window.pageYOffset* і *window.pageXOffset*. Їх можна тільки читати, а міняти не можна. Але ці властивості не підтримуються IE8.

Крім того, в IE7- <html> зміщений відносно (0,0). Тому крос-браузерний варіант отримання прокрутки сторінки по вертикалі з урахуванням браузера IE8 матиме такий вигляд*:*

```
var html = document.documentElement;
var body = document.body;

var scrollTop = html.scrollTop || body && body.scrollTop
|| 0;
scrollTop -= html.clientTop;
```

Аналогічно, використовуючи властивість *scrollLeft*, можна отримати величину прокрутки сторінки по горизонталі.

Для того, щоб дізнатися, чи переглянув користувач до кінця, елемент, наприклад сторінку, що має прокручування, необхідно виконати таке порівняння:

```
(html.scrollHeight - html.scrollTop ===
                                    html.clientHeight)
```

Якщо воно повертає true, то прокрутка виконана повністю, інакше – ні. Об'єкти інших елементів, які можуть мати прокручування, теж підтримують властивості, що використовуються в цьому виразі.

Властивість *scrollHeight* (тільки читання) – це висота контенту в елементі (у пікселях), включаючи вміст, невидимий через прокручування.

Властивість *scrollTop* – кількість пікселів, прокручених від верху елемента.

Властивість *clientHeight* (тільки читання) – це фактична висота в пікселях частини елемента, видимої користувачеві (одного екрана).

6.3.2.2 Виконання прокрутки сторінки

Щоб прокрутити сторінку за допомогою JavaScript, її DOM повинен бути повністю завантажений.

На звичайних елементах властивості *scrollTop / scrollLeft* можна змінювати, і при цьому елемент буде прокручуватися. Але для сторінки ці властивості не працюють в деяких браузерах.

Для прокрутки сторінки можна використовувати спеціальні методи:

window.scrollBy(x,y) – прокручує сторінку відносно поточних координат;

window.scrollTo(pageX,pageY) – прокручує сторінку до зазначених координат відносно документа.

Встановити прокручування елемента *elem* таким чином, щоб він став видимим користувачеві, можна за допомогою такого методу:

```
elem.scrollIntoView(alignToTop|scrollIntoViewOptions)
```

Аргумент (необов'язковий) може бути або булевого типу (*alignToTop*), або об'єктом (*scrollIntoViewOptions*).

Значення *true* (діє за замовчуванням) задає установку верхньої межі елемента вгорі видимої частини вікна прокручуваної області.

Значення *false* задає установку нижньої межі елемента внизу видимої частини вікна прокручуваної області.

Об'єкт може містити такі властивості:

behavior – визначає анімацію скролла;

block – визначає вертикальне вирівнювання;

inline – визначає горизонтальне вирівнювання.

6.3.2.3 Подія прокрутки

Подія прокрутки *scroll* дозволяє реагувати на прокрутку сторінки або елемента. Дана подія є спливаючей.

Цю подію можна використовувати для виконання багатьох корисних дій, наприклад:

- показати / приховати додаткові елементи управління або інформацію, грунтуючись на тому, в якій частині документа знаходиться користувач;

- довантажити дані, коли користувач прокручує сторінку вниз до кінця.
- довантажити і ініціалізувати елементи інтерфейсу, які стали видимими після прокрутки.

6.4 Виконання завдання

6.4.1 Практичне заняття виконується в рамках веб-сайту, визначеного в якості індивідуального завдання в ПЗ № 1.

6.4.2 Отримайте індивідуальне завдання у викладача.

6.4.3 Відповідно до отриманого завдання додайте нову сторінку (при необхідності) або перетворіть існуючі сторінки, розробіть необхідний HTML-, CSS- і JS-код.

6.4.4 Продемонструйте результати роботи викладачеві в браузері.

4.4.5 Складіть звіт, оформіть його відповідно до ДСТУ 3008: 2015 [4] і захистите роботу.

6.4.6 Звіт повинен містити:

– номер і суть індивідуального завдання, найменування сайту;
– HTML-код, CSS-код і JS-код всіх розроблених і змінених сторінок;
– скріншоти вікон браузера, що демонструють результати виконання завдання.

6.5 Контрольні завдання

Для всіх варіантів завдання для зберігання даних на комп'ютері користувача необхідно використовувати локальне сховище Web Storage.

Перед збереженням даних необхідно перевірити, чи підтримується Web Storage браузером, і запросити дозвіл у користувача на використання локального сховища, використавши, наприклад, метод `confirm()` об'єкта `Window`.

Перелік індивідуальних завдань дивиться в окремому файлі.

6.6 Контрольні запитання

1. З якою метою може виникнути необхідність зберігати дані на стороні клієнта?

2. Які недоліки використання Cookies для зберігання інформації на стороні клієнта?

3. Які переваги має зберігання даних на стороні клієнта за допомогою Web Storage в порівнянні з Cookies?

4. В якому вигляді зберігаються дані в веб-сховище?

5. Чим відрізняється веб-сховище локального типу з веб-сховища даних сеансу?

6. Які існують способи збереження даних в веб-сховище?

7. Чи можна дані веб-сховища, збережені однією сторінкою, прочитати з іншої сторінки сайту?

8. Яке призначення підоб'єкту *SessionStorage* об'єкта *Window*?

9. Що містить властивість *length* об'єкта *LocalStorage*?

10. У якому випадку і де виникає подія storage?

11. Як можна використовувати подію *storage* для обміну повідомленнями між вікнами одного сайту?

12. Які існують особливості зберігання нестрокових даних в веб-сховище?

13. Який об'єкт використовується при читанні дат з веб-сховища?

14. Яке призначення методу *JSON.stringify*?

15. Яке призначення методу *window.scrollTo (pageX, pageY)*?

16. У якому випадку виникає подія *scroll*?

17. Для чого можна використовувати подію *scroll?*

ПЕРЕЛІК ПОСИЛАНЬ

1. Leading the web to its full potential [Электронный ресурс] / W3C. – Режим доступу: www/ URL: http://www.w3.org/ -– Загол. з екрану.

2. Tutorial HTML [Электронный ресурс] / W3schools. – Режим доступу: www/ URL: https://www.w3schools.com/ -– Загол. з екрану.

4. ДСТУ 3008:2015 "Звіти у сфері науки і техніки. Структура та правила оформлення".

Електронне навчальне видання

Методичні вказівки

до практичних занять

з дисципліни «WEB-ТЕХНОЛОГІІ ТА WEB-ДИЗАЙН»

для студентів усіх форм навчання
спеціальності 122 «Комп'ютерні науки»

Упорядники: БОРИСЕНКО Тетяна Іванівна

Відповідальний випусковий К.Е. Петров

Редактор

Комп'ютерна верстка

План 20__ (_____ півріччя) , поз. ____
Підп. до використання ___.___.20___. Формат pdf Обсяг даних ___ Мб.

ХНУРЕ, 61166, Харків, просп. Науки, 14, E-mail: info@nure.ua

Підготовлено в редакційно-видавничому відділі ХНУРЕ
Свідоцтво суб'єкта видавничої справи ДК №1409 від 26.06.2003