**Практична робота № 4**

**Тема**. Робота з Document Object Model (DOM) в JavaScript

**Мета**: ознайомлення та набуття практичних навичок написання JS-скриптів з використанням DOM для взаємодії з елементами сторінки.

**Короткі теоретичні відомості**

**Об’єктна модель документа** (DOM**)** з’єднує веб-сторінки зі сценаріями або мовами програмування, представляючи структуру документа, наприклад HTML, що представляє веб-сторінку, у пам’яті. Зазвичай це стосується JavaScript, навіть якщо моделювання документів HTML, SVG або XML як об’єктів не є частиною основної мови JavaScript.

Об’єкт document – це головна “точка входу” до сторінки. Ми можемо змінити або створити що-небудь на сторінці, використовуючи цей об’єкт.

Наприклад

// змінити колір фону на червоний

document.body.style.background = "red";

// повернути його назад після 1 секунди

setTimeout(() => document.body.style.background = "", 1000);

Тут ми використовували document.body.style, але там набагато, набагато більше властивостей. Властивості та методи описані у специфікації: DOM Living Standard.

Вам не потрібно робити нічого особливого, щоб почати використовувати DOM. Ви використовуєте API безпосередньо в JavaScript із того, що називається скриптом, який запускає браузер.

Коли ви створюєте скрипт, вбудований в елемент <script> або включений у веб-сторінку, ви можете негайно почати використовувати API для об’єктів документа чи вікна для керування самим документом або будь-яким із різноманітних елементів веб-сторінки (нащадкові елементи документа).

Нижче наведено ще один простий спосіб роботи з DOM, який відображає повідомлення на консолі за допомогою функції console.log():

<body onload="console.log('Welcome to my home page!');">

  …

</body>

**Подія** – це сигнал від браузера, що *щось* сталося. Всі DOM-вузли подають такі сигнали (хоча події бувають не тільки в DOM).

Ось список найпоширеніших DOM-подій для ознайомлення:

*Події миші:*

click – відбувається, коли користувач натискає елемент лівою кнопкою миші (на пристроях із сенсорними екранами воно відбувається при торканні).

contextmenu – відбувається, коли клацнули на елемент правою кнопкою миші.

mouseover / mouseout – коли миша наводиться на / залишає елемент.

mousedown / mouseup – коли натиснули / відпустили кнопку миші на елементі.

mousemove – під час руху миші.

*Події клавіатури:*

keydown та keyup – коли користувач натискає / відпускає клавішу.

*Події елементів форми*:

submit – користувач надіслав форму <form>.

focus – користувач фокусується на елементі, наприклад, натискає на <input>.

*Події документа*:

DOMContentLoaded – коли HTML завантажено й оброблено, DOM документа повністю побудований і доступний.

*CSS події*:

transitionend – коли CSS-анімацію завершено.

Існує багато інших подій.

Для подій можна призначити обробника, тобто функцію, яка спрацює, щойно подія сталася.

Саме завдяки обробникам JavaScript код може реагувати на дії користувача.

Є кілька способів призначити обробника події.

*Використання атрибута HTML*

Обробник може бути призначений прямо в розмітці, атрибуті, який називається on<event>.

Наприклад, щоб призначити обробник події click на елементі input, можна використовувати атрибут onclick, ось так:

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

При натисканні мишкою на кнопці виконається код, вказаний в атрибуті onclick.

*Використання властивостей DOM-об’єкта*

Можемо призначати обробник, використовуючи властивість DOM-елемента on<event>.

Наприклад, elem.onclick:

    <input id="elem" type="button" value="Click me">

    <script>

        elem.onclick = function () {

            alert('Дякую');

        };

    </script>

Якщо обробник заданий через атрибут, то браузер читає HTML-розмітку, створює нову функцію із вмісту атрибута та записує у властивість.

Цей спосіб, по суті, аналогічний до попереднього.

*Використання addEventListener*

Фундаментальний недолік описаних вище способів присвоєння обробника – неможливість повісити кілька обробників для однієї події.

Наприклад, одна частина коду хоче при натисканні на кнопку підсвітити її, а інша – показати повідомлення.

Ми хочемо призначити два обробники для цього. Але новий обробник перезапише попередній:

input.onclick = function() { alert(1); }

// ...

input.onclick = function() { alert(2); } // замінить попередній обробник

Розробники стандартів досить давно це зрозуміли і запропонували альтернативний спосіб призначення обробників за допомогою спеціальних методів addEventListener та removeEventListener. Вони вільні від цього недоліку.

Синтаксис додавання обробника:

element.addEventListener(event, handler, [options]);

event – назва події, наприклад ‘click’.

handler – посилання на функцію-обробник.

options – додатковий об’єкт із властивостями:

* once: якщо true, тоді обробник буде автоматично вилучений після виконання.
* capture: фаза, на якій повинен спрацювати обробник, докладніше про це розказано у розділі [*Bubbling and capturing*](https://uk.javascript.info/bubbling-and-capturing). Так історично склалося, що options може бути false/true, це те саме, що {capture: false/true}.

passive: якщо true, тоді обробник ніколи не викличе preventDefault(), докладніше про це розказано у розділі [Типові дії браузера](https://uk.javascript.info/default-browser-action).

Для видалення обробника слід використовувати removeEventListener:

element.removeEventListener(event, handler, [options]);

Властивості навігації по DOM чудові, коли елементи розташовані близько один до одного. Але якщо є способи отримати довільний елемент чи елементи сторінки. Одним з таких способів є document.getElementById

Якщо елемент має атрибут id, ми можемо отримати його за допомогою методу document.getElementById(id), незалежно від того, де він знаходиться.

Наприклад:

    <div id="elem">

        <div id="elem-content">Елемент</div>

    </div>

    <script>

        // отримати елемент

        let elem = document.getElementById('elem');

        // зробити його фон червоним

        elem.style.background = 'red';

    </script>

Важливо, щоб id був унікальним на сторінці, тобто у документі може бути лише один елемент із заданим id.

Якщо є кілька елементів з однаковим id, то поведінка методів, які його використовують, буде непередбачуваною, наприклад. document.getElementById може повертати будь-який з таких елементів випадковим чином. Тому, будь ласка, дотримуйтеся правила та залишайте id унікальним.

Метод getElementById можна викликати лише для об’єкта document. Він шукає вказаний id у всьому документі.

**Додаткові матеріали:**

1. <https://uk.javascript.info/browser-environment>
2. <https://uk.javascript.info/dom-nodes>
3. <https://uk.javascript.info/dom-navigation>
4. <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction>
5. <https://uk.javascript.info/introduction-browser-events>

**Прототип**

Прототип можна скачати на GitHub: <https://github.com/mchesanovskyy/js_sets/tree/main/prototypes/practice4>

**Лістинг «index.html»**

<!DOCTYPE html>

<html>

<head>

    <title>Практична робота № 4 (ПІБ)</title>

    <meta charset="UTF-8">

    <link rel="stylesheet" href="practice4.css" />

    <script defer src="practice4.js"></script>

</head>

<body>

    <div class="container">

        <div class="row">

            <label>Прізвище:</label>

            <input id="txtLastName" />

        </div>

        <div class="row">

            <label>Ім'я:</label>

            <input id="txtFirstName"/>

        </div>

        <div class="row">

            <label>По батькові:</label>

            <input id="txtMiddleName" />

        </div>

        <div class="row">

            <label>Телефон:</label>

            <input id="txtEmail" />

        </div>

        <div class="row row-reverse">

            <input id="btnSubmit" class="btn" type="button" value="Submit" />

        </div>

        <div class="row hidden">

            <textarea id="txtResult" cols="35" rows="5"></textarea>

            <button id="btnClose" class="btn">Close</button>

        </div>

    </div>

</body>

</html>

**Лістинг «practice4.css»**

.container {

    background-color: #7b9eaf;

    padding: 15px 10px;

    border-radius: 8px;

    margin: 10px auto;

    width: 300px;

    display: flex;

    flex-direction: column;

}

.row {

    margin-top: 10px;

    display: flex;

    flex-direction: row;

    justify-content: space-between;

}

.row label {

    flex-shrink: 3;

}

.row-reverse {

    justify-content: flex-end;

}

textarea {

    width: 100%;

}

.hidden {

    display: none;

}

.btn {

    cursor: pointer;

    margin-left: 10px;

}

**Лістинг «practice4.js»**

// шукаємо елемент, де id=btnSubmit

let submitButton = document.getElementById('btnSubmit');

// підписуємо функцію btnSubmitOnClick на подію 'click'

// для елементу btnSubmit.

submitButton.addEventListener("click", btnSubmitOnClick);

// Функція - ізольована частина коду, яка виконуватиметься

// тільки при зверненні до неї за назвою.

// Синтаксис виклику ф-ції: btnSubmitOnClick();

// В цьому випадку код з ф-ції викликатиметься кожного разу

// при настисканні на кнопку.

function btnSubmitOnClick() {

     // **todo:** поле для виконання практичної роботи

}

**Порядок виконання роботи**

Створіть html-, js-, css- файли, додайте прототип коду та доповніть код згідно наведених нижче пунктів.

1. Отримайте введені значення текстових полів для елементів Прізвище, Ім’я, По батькові та номер телефону і збережіть їх в окремі змінні.
   1. Використовуйте document.getElementById(‘id-attr’) для отриманні елементу
   2. Використовуйте властивість value (напр. firstNameElem.value) для отримання поточного введеного значення.
2. Створіть змінну info, яку сформуйте в наступному форматі

Прізвище: <value>

Ім'я: <value>

По батькові: <value>

Телефон: <value>

1. Виведіть значення змінної info на сторінці в елементі з txtResult.
2. Оскільки елемент txtResult знаходиться в невидимому div блоці через присутній там клас hidden (<div class="row hidden">), потрібно:
   1. Знайти цей div елемент. Для цього можна скористатись функцією closest() чи parentElement відносно елементу txtResult або додати значення id для цього елементу, щоб отримати його.
   2. Видалити клас hidden. Для цього можна скористатись функцією remove() для властивості classList.

elem.classList.remove('hidden')

1. Підпишіть кнопку btnClose на подію ‘click’, щоб при натисканні відбувались наступні дії:
   1. Очищувалось значення елементу txtResult. Можна присвоїти порожнє значення.
   2. Додавався клас hidden для блоку результатів, щоб той знову став невидимим. Для цього можна скористатись функцією add() для властивості classList.

elem.classList.add('hidden')

**Приклад виконання програми**

|  |  |
| --- | --- |
| 1. Вигляд при запуску сторінки | 1. Вигляд сторінки після натискання кнопки «Submit» |
| 1. Вигляд сторінки після натискання кнопки «Close» |  |

У звіті представити **кінцевий** варіант коду та знімки екрану з результатами роботи.