**Лабораторна 2**

**Вступ до подій браузера**

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, поки що просто для ознайомлення:

**Події миші:**

* click – відбувається, коли клацнули на 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.

**[Обробники подій](https://uk.javascript.info/introduction-browser-events" \l "obrobniki-podii)**

Події можна призначити обробник, тобто функцію, яка спрацює, щойно подія сталася.

Саме завдяки обробникам JavaScript код може реагувати на дії користувача.

Є кілька способів призначити події обробника.

**[Використання атрибута HTML](https://uk.javascript.info/introduction-browser-events" \l "vikoristannya-atributa-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.

Зверніть увагу, що для вмісту атрибуту onclick використовуються одинарні лапки, а сам атрибут знаходиться в подвійних. Якщо ми забудемо про це і поставимо подвійні лапки всередині атрибуту, ось так: onclick="alert("Click!")", код не буде працювати.

Атрибут HTML-тега – не найзручніше місце для написання великої кількості коду, тому краще створити окрему JavaScript-функцію та викликати її там.

Наступний приклад по кліку запускає функцію countRabbits():

<script>

function countRabbits() {

for(let i=1; i<=3; i++) {

alert("Кролик номер " + i);

}

}

</script>

<input type="button" onclick="countRabbits()" value="Рахувати кроликів!">

Як ми пам’ятаємо, атрибут HTML-тега не чутливий до регістру, тому ONCLICK буде працювати так само як onClick і onCLICK… Але, як правило, атрибути пишуть у нижньому регістрі onclick.

**[Використання властивостей DOM-об’єкта](https://uk.javascript.info/introduction-browser-events" \l "vikoristannya-vlastivostei-dom-obyekta)**

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.

Ці два приклади коду працюють однаково:

Тільки HTML:

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

HTML + JS:

<input type="button" id="button" value="Кнопка">

<script>

button.onclick = function() {

alert('Клік!');

};

</script>

У першому прикладі використовувався атрибут HTML для ініціалізації button.onclick, тоді як у другому прикладі – сценарій, ось і вся різниця.

**Оскільки в елемента DOM може бути тільки одна властивість з ім’ям onclick, то призначити більше одного обробника таким чином не можна.**

У прикладі нижче призначення обробника додатково через JavaScript перезапише обробник з атрибуту:

<input type="button" id="elem" onclick="alert('Було')" value="Click me">

<script>

elem.onclick = function() { // Перезапише існуючий обробник

alert('Стало'); // виведеться лише це повідомлення

};

</script>

Щоб видалити обробник – установіть elem.onclick = null.

**[Доступ до елемента через this](https://uk.javascript.info/introduction-browser-events" \l "dostup-do-elementa-cherez-this)**

Усередині обробника події this посилається на поточний елемент, тобто на той, на якому, як кажуть, «висить» (тобто призначений) обробник.

У коді нижче button виводить свій вміст, використовуючи this.innerHTML:

<button onclick="alert(this.innerHTML)">Натисни на мене</button>

**[Поширені помилки](https://uk.javascript.info/introduction-browser-events" \l "poshireni-pomilki)**

Якщо ви починаєте працювати з подіями, зверніть увагу на наступні моменти.

Ви можете призначити функцію що вже визначена, як обробника:

function sayThanks() {

alert('Thanks!');

}

elem.onclick = sayThanks;

Будьте обережні! Функція повинна бути присвоєна в такому вигляді – sayThanks, а не sayThanks().

// правильно

button.onclick = sayThanks;

// неправильно

button.onclick = sayThanks();

Якщо додати дужки, то sayThanks() – це вже виклик функції, результат якого (рівний undefined, тому що функція нічого не повертає) буде присвоєно onclick. Тож це не буде працювати.

…А ось у розмітці, на відміну від властивості, дужки потрібні:

<input type="button" id="button" onclick="sayThanks()">

Цю різницю просто пояснити. При створенні обробника браузером з атрибута він автоматично створює функцію з тілом зі значення атрибута:

Таким чином розмітка генерує таку властивість:

button.onclick = function() {

sayThanks(); // <-- вміст атрибуту

};

**Не використовуйте setAttribute для обробників.**

Такий виклик не буде працювати:

// При натисканні на <body> виникнуть помилки,

// атрибути завжди рядки, і функція стане рядком

document.body.setAttribute('onclick', function() { alert(1) });

**Регістр DOM-властивості має значення…**

Використовуйте elem.onclick, а не elem.ONCLICK, тому що DOM-властивості чутливі до регістру.

**[addEventListener](https://uk.javascript.info/introduction-browser-events" \l "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. 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()

Для видалення обробника слід використовувати removeEventListener:

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

**Видалення підписника на подію вимагає саме ту ж функцію**

Для видалення потрібно передати саме ту функцію-обробник, яка була присвоєна.

Отак не спрацює:

elem.addEventListener( "click" , () => alert('Дякую!'));

// ....

elem.removeEventListener( "click", () => alert('Дякую!'));

Обробник не буде видалено, так як removeEventListener передано не таку ж функцію, а іншу, з однаковим кодом.

Ось так вірно:

function handler() {

alert( 'Дякую!' );

}

input.addEventListener("click", handler);

// ....

input.removeEventListener("click", handler);

Зверніть увагу – якщо функцію обробник не зберегти будь-де, ми не зможемо її видалити. Немає методу, який дозволяє отримати з елемента обробники подій, присвоєні через addEventListener.

Метод addEventListener дозволяє додавати кілька обробників на одну подію одного елемента, наприклад:

<input id="elem" type="button" value="Натисни мене"/>

<script>

function handler1() {

alert('Дякую!');

};

function handler2() {

alert('Ще раз дякую!');

}

elem.onclick = () => alert("Привіт");

elem.addEventListener("click", handler1); // Дякую!

elem.addEventListener("click", handler2); // Ще раз дякую!

</script>

Як видно з прикладу вище, можна *одночасно* призначати обробники через DOM-властивість і через addEventListener. Однак, щоб уникнути плутанини, рекомендується вибрати один спосіб.

**Обробники деяких подій можна присвоїти лише через addEventListener**

Існують події, які не можна призначити через DOM-властивість, але можна через addEventListener.

Наприклад, така подія DOMContentLoaded, яке спрацьовує, коли завершено завантаження та побудову DOM документа.

// не буде працювати

document.onDOMContentLoaded = function() {

alert("DOM побудований");

};

// буде працювати

document.addEventListener("DOMContentLoaded", function() {

alert("DOM побудований");

});

Таким чином addEventListener більш універсальний. Хоча зауважимо, що таких подій меншість, це скоріше виняток, ніж правило.

**[Об’єкт події](https://uk.javascript.info/introduction-browser-events" \l "obyekt-podiyi)**

Щоб правильно обробити подію, можуть знадобитися деталі того, що сталося. Не просто “клік” або “натискання клавіші”, але й координати вказівника миші, яка саме клавіша натиснута і так далі.

Коли відбувається подія, браузер створює *об’єкт події*, записує в нього деталі та передає його як аргумент функції-обробнику.

Приклад нижче демонструє отримання координат миші з події:

<input type="button" value="Натисни мене" id="elem">

<script>

elem.onclick = function(event) {

// вивести тип події, елемент та координати кліка

alert(event.type + " на " + event.currentTarget);

alert("Координати: " + event.clientX + ":" + event.clientY);

};

</script>

Деякі властивості об’єкту event:

**event.type**

Тип події, у цьому випадку "click".

**event.currentTarget**

Елемент, у якому спрацював обробник. Це значення зазвичай таке саме, як і в this, але якщо обробник є функцією-стрілкою чи за допомогою bind прив’язаний інший об’єкт this, то ми можемо отримати елемент з event.currentTarget.

**event.clientX / event.clientY**

Координати курсору в момент кліку у площині вікна для подій миші.

Існує багато властивостей подій. Більшість з них залежать від типу події: події клавіатури мають один набір подій, а події вказівника – інший. Ми розглянемо детальніше деякі з них пізніше.

**Об’єкт події доступний і в HTML**

При призначенні обробника в HTML теж можна використовувати об’єкт event, ось так:

<input type="button" onclick="alert(event.type)" value="Тип події">

Це можливо тому, що коли браузер з атрибуту створює функцію-обробник, вона виглядає так: function(event) { alert(event.type) }. Тобто, її перший аргумент називається "event", а тіло взяте з атрибуту.

**[Об’єкт-обробник: handleEvent](https://uk.javascript.info/introduction-browser-events" \l "obyekt-obrobnik-handleevent)**

Ми можемо призначити обробником не лише функцію, а й об’єкт за допомогою addEventListener. У такому разі, коли відбувається подія, викликається метод об’єкта handleEvent.

Наприклад:

<button id="elem">Натисни мене</button>

<script>

let obj = {

handleEvent(event) {

alert(event.type + " at " + event.currentTarget);

}

};

elem.addEventListener('click', obj);

</script>

Як бачимо, якщо addEventListener отримує об’єкт як обробник, він викликає obj.handleEvent(event), коли відбувається подія.

Ми також можемо використати клас для цього, ось так:

<button id="elem">Натисни мене</button>

<script>

class Menu {

handleEvent(event) {

switch(event.type) {

case 'mousedown':

elem.innerHTML = "Натиснута кнопка миші";

break;

case 'mouseup':

elem.innerHTML += "...і відпущена.";

break;

}

}

}

let menu = new Menu();

elem.addEventListener('mousedown', menu);

elem.addEventListener('mouseup', menu);

</script>

Тут той самий об’єкт обробляє обидві події. Зверніть увагу, ми повинні явно призначити обидва обробники через addEventListener. Тоді об’єкт menu отримуватиме події mousedown та mouseup, а не інші (не призначені) типи подій.

Метод handleEvent не обов’язково має виконувати всю роботу сам. Він може викликати інші методи, які створені під обробку конкретних типів подій, ось так:

<button id="elem">Натисни мене</button>

<script>

class Menu {

handleEvent(event) {

// mousedown -> onMousedown

let method = 'on' + event.type[0].toUpperCase() + event.type.slice(1);

this[method](event);

}

onMousedown() {

elem.innerHTML = "Натиснута кнопка миші";

}

onMouseup() {

elem.innerHTML += "...і відпущена.";

}

}

let menu = new Menu();

elem.addEventListener('mousedown', menu);

elem.addEventListener('mouseup', menu);

</script>

Тепер обробка подій розділена методами, що спрощує підтримку коду.

**Завдання**

1. Створити випадаюче меню на своєму сайті.
2. Пункти і підпункти меню мають бути робочі, тобто при натисканні на них переходити на відповідні сторінки.
3. Меню має бути написане на HTMl+CSS+ Javascript, без додаткових бібліотек та фреймворків

**Приклад меню**

Створимо структуру меню у HTML коді. Нехай все меню буде представлено блоком з id=«nav». Усередині блоку містяться блоки, що відповідають за пункти меню, їм присвоєно class=«menu-item». Сам пункт меню не виділений тегами, але при необхідності ви маєте можливість його обернути та оформити на його смак. Меню, що випадають, закріплені за кожним пунктом, знаходяться в блоках class=«submenu». Весь код представимо наступним чином:

<div id="nav">

        <div class="menu-item">

            Меню 1

            <div class="submenu">

                Подменю

            </div>

        </div>

        <div class="menu-item">

            Меню 2

            <div class="submenu">

                Подменю

            </div>

        </div>

        <div class="menu-item">

            Меню 3

            <div class="submenu">

                Подменю

            </div>

        </div>

    </div>

У файлі style.css вкажемо такі стилі:

/\*Для пунктів меню\*/

.menu-item {

cursor: pointer; /\*Вказівник курсору миші\*/

}

Для меню, що розкриваються, задаємо наступні стилі:

.menu-item .submenu {

display: none; /\*приховуємо всі пункти меню\*/

cursor: pointer;

}

Відображати меню потрібно при наведенні на пункт меню курсору миші (подія **onmouseover**). Дану подію ми відстежуватимемо тільки всередині блоку з id=«nav»

document.getElementById('nav').onmouseover= function(event) {

}

Цей запис відстежує знаходження миші всередині блоку навігації. Потім, необхідно з'ясувати де було зроблено клік мишею. Для цього ми заведемо змінну target куди занесемо event.target - поточний елемент, на якому сталася подія.

document.getElementById('nav').onmouseover= function(event) {

var target = event.target; // де було натискання?

}

Тепер потрібно перевірити, щоб той блок, на якому зараз знаходиться миша, мав клас menu-item. Якщо це так, необхідно отримати блок з class=«submenu» всередині нього. Це можна зробити за допомогою команди var s=target.getElementsByClassName(«submenu»). В результаті роботи команди в змінній s знаходитиметься масив з одного елемента, що містить вкладене меню. Звертатися щодо нього можна з допомогою ідентифікатора s[0].

Після того, як ми з'ясували, де знаходиться миша і який блок підменю потрібно відобразити, виконаємо дії: приховаємо всі відкриті меню і відобразимо поточне підменю:

document.getElementById('nav').onmouseover= function(event) {

var target = event.target; // де було натискання?

if (target.className == 'menu-item') {

var s=target.getElementsByClassName('submenu');

closeMenu();

s[0].style.display='block';

}

}

Приховувати всі відкриті меню ми за допомогою функції closeMenu(). Ця функція отримає всі елементи submenu в блоці nav і за допомогою циклу всім надає стилі style.display=«none» . Повний код функції:

function closeMenu(){

var menu=document.getElementById('nav');

var subm=document.getElementsByClassName('submenu');

for (var i=0; i<subm.length; i++) {

subm[i].style.display="none";

}

}

Меню потрібно приховувати, якщо миша пішла за межі блоку nav. Однак, поки миша знаходиться на підменю, її потрібно відображати. Запишемо функцію, яка відстежуватиме положення миші та перевірятиме елементи під курсором. Якщо ці елементи не є menu-item або submenu, то буде закривати всі відкриті меню.

document.onmousemove=function(event) {

var target = event.target; // где был клик?

console.log(event.target);

if (target.className!='menu-item' && target.className!='submenu') {

closeMenu();

}

}