10.6. JavaScript у веб-розробці

Моделі BOM та DOM. Навігація за елементами документа. Доступ до вузлів та атрибутів сторінки. Зміна структури DOM. Основні браузерні події

1. Браузерне оточення
    a. Об'єкт window
    b. Об'єкт document - модель DOM
    c. Об'єктна модель браузера – BOM
    d. Підключення скриптів до сайту
2. Навігація по DOM
3. Властивості DOM-вузлів
4. Пошук по дереву документ
5. Внесення змін до DOM-дерева
    a. Створимо текстовий елемент та заголовок першого рівня на початку сторінки .
    b. Копіювання абзацу та додавання його після секції із заміною тексту.
    c. Замінити всі теги на сторінці заголовками другого рівня.
    d. Додавання тексту із спеціальними символами
6. Управління стилями елементів
    a. Виведення вмісту атрибута "class".
    b. Зміна класів
    c. Зміна атрибутів тегів
    d. Скидання стилів
7. Браузерні події
    a. Обробка події через властивість тегу.
    b. Обробка події через вибір елемента у js-файлі.
    c. Обробка події через спеціальні методи.
    d. Параметр event функції-обробника.
    e. Етапи події (занурення, досягнення мети, спливання).
8. Делегування подій

Браузерне оточення

Мова JavaScript створювалася, в першу чергу, для роботи з фронтендом, клієнтською стороною інтерфейсу користувача. З його допомогою HTML та CSS стають максимально керованими, змінюваними, з гарантією роботи практично на всіх пристроях.

Сьогодні CSS оснащений функціоналом, здатним замінити частину коду JavaScript, але цього все ще недостатньо, та й досягається значно більшими зусиллями. JS-скрипти додають динаміку на веб-сторінки, роблячи їх інтерфейс гранично ергономічним та зручним для всіх типів користувачів.

Можливості JavaScript вже не обмежені тільки фронтендом, його застосовують на серверній стороні, різних платформах та пристроях. Наше завдання: розглянути цю мову програмування в плані її використання в браузері, на стороні користувача.

Щоб JavaScript отримав можливість запуску (у браузері, операційній системі) необхідно надати йому певну функціональність, так зване називають оточення. Оточення дозволяє мові програмування отримати доступ до своїх об'єктів, функцій, змінних на додаток до базових

інструментів JavaScript. Зокрема, браузер «дозволяє» керувати веб-сторінками. Схема браузерного оточення наведена нижче:

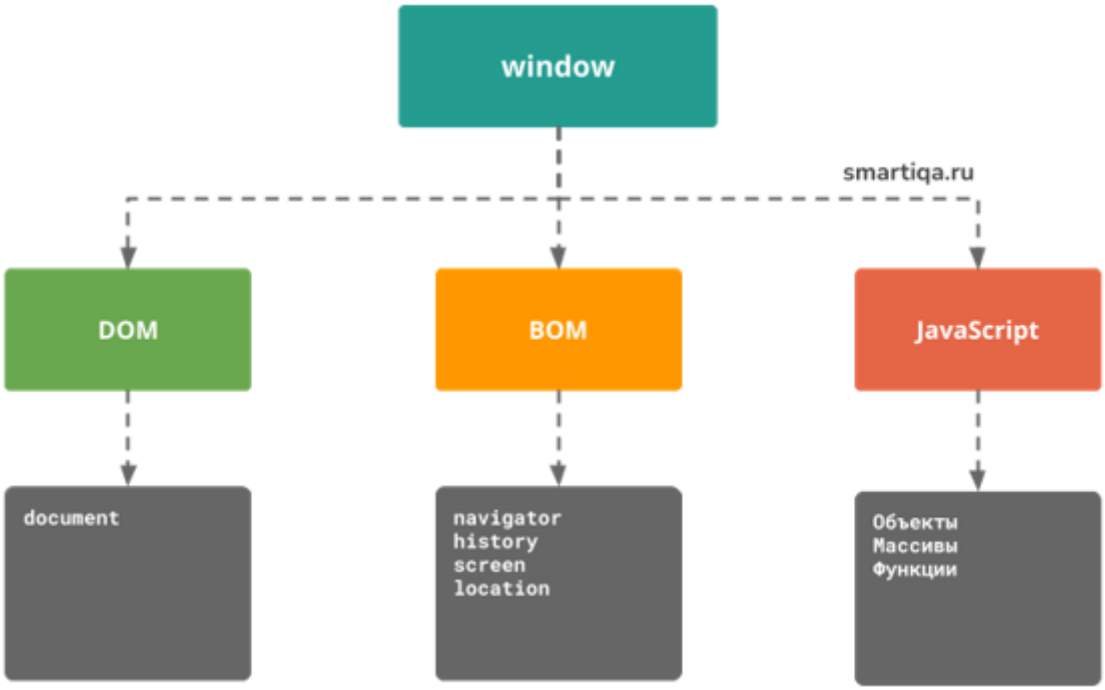


Рис. Об'єкти браузерного оточення, доступні для мови JavaScript

Окрім можливостей самої мови, отримуємо доступ до ряду об'єктів:

- Об'єкт window. Кореневий елемент, що є вікном браузера і містить методи управління ним.
- Об'єкт DOM (Document Object Model, об'єктна модель документа) - сукупність всіх сутностей веб-сторінки у вигляді дерева.
- Об'єкт BOM (Browser Object Model, об'єктна модель браузера) – додаткові інструменти для безпосередньої роботи з браузером.

Об'єкт window

Дозволяє звертатися до змінних та функцій у будь-якому місці програми. Ключове слово window можна опускати.

Приклад - Консоль браузера

```
> window.prompt('Готові до JS?')
< так
> prompt('Готові до JS?')
< null
```

Якщо в Google Chrome відкрити Панель Інструментів розробника і вибрати вкладку Console , можна писати js-код та оперувати з об'єктами. У прикладі вище в першому випадку ми звернулися до функції prompt() і ввели так, у другому варіанті ми натиснули на скасування, що повернуло null.

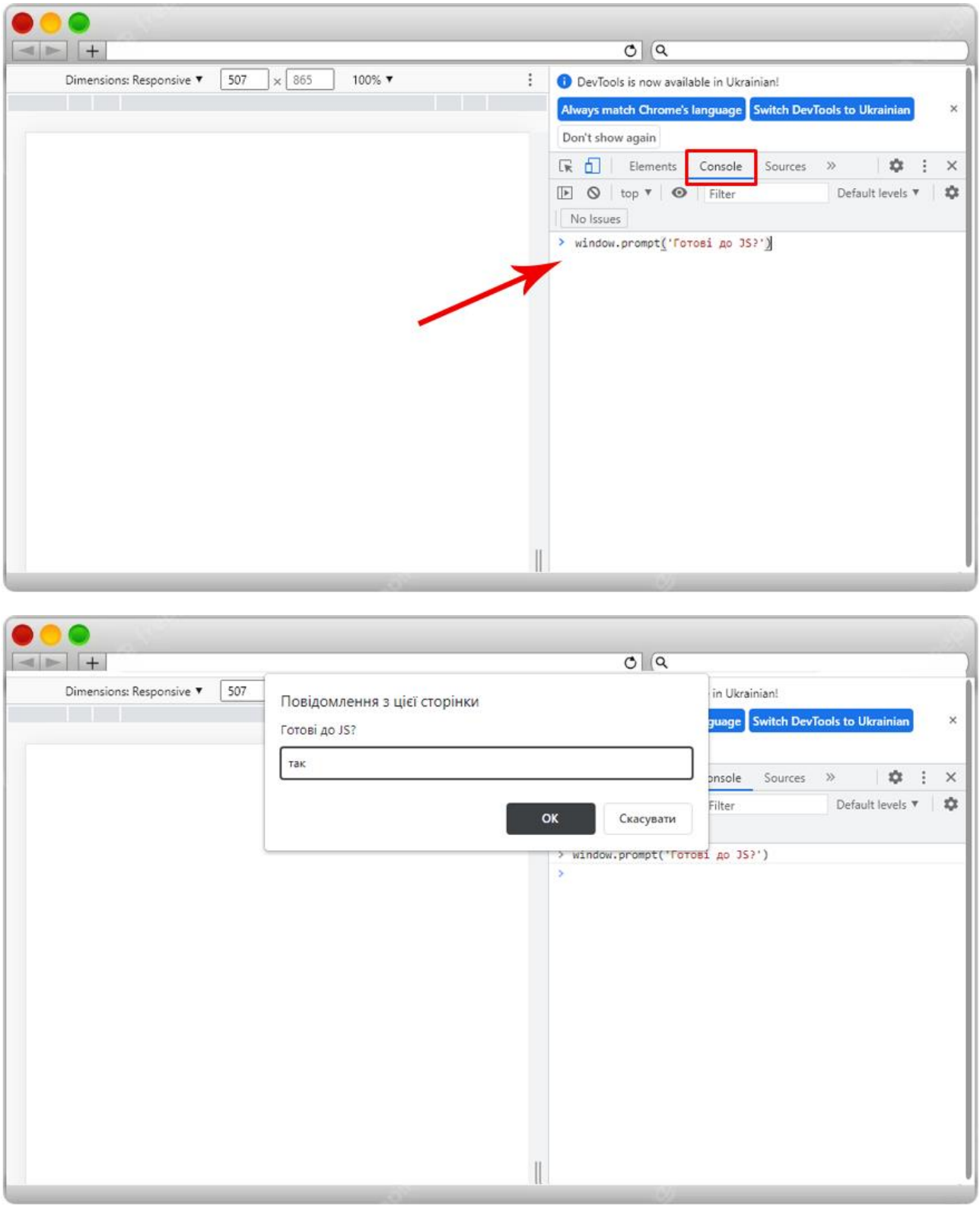


Рис. Виклик методу prompt() із консолі браузера

Оскільки у вікон, що виникають в результаті використання методів prompt(), alert(), confirm() немає можливостей модифікування, їх практичне застосування не вітається.

Перелік всіх властивостей та методів об'єкта window величезний (побачити можна, якщо ввести в консолі window) - властивості performance, screen, методи back(), find() та інші.

Об'єкт document - модель DOM

Потрібен для створення та зміни елементів на сторінці. Породжує так звану модель DOM . Усі теги, коментарі та текст веб-ресурсу стають об'єктами, з якими можна працювати.

Приклад - Консоль браузера

```
> document.body.style.fontSize="30px";
```

За допомогою цієї команди ми змінюємо розмір шрифту сторінки.

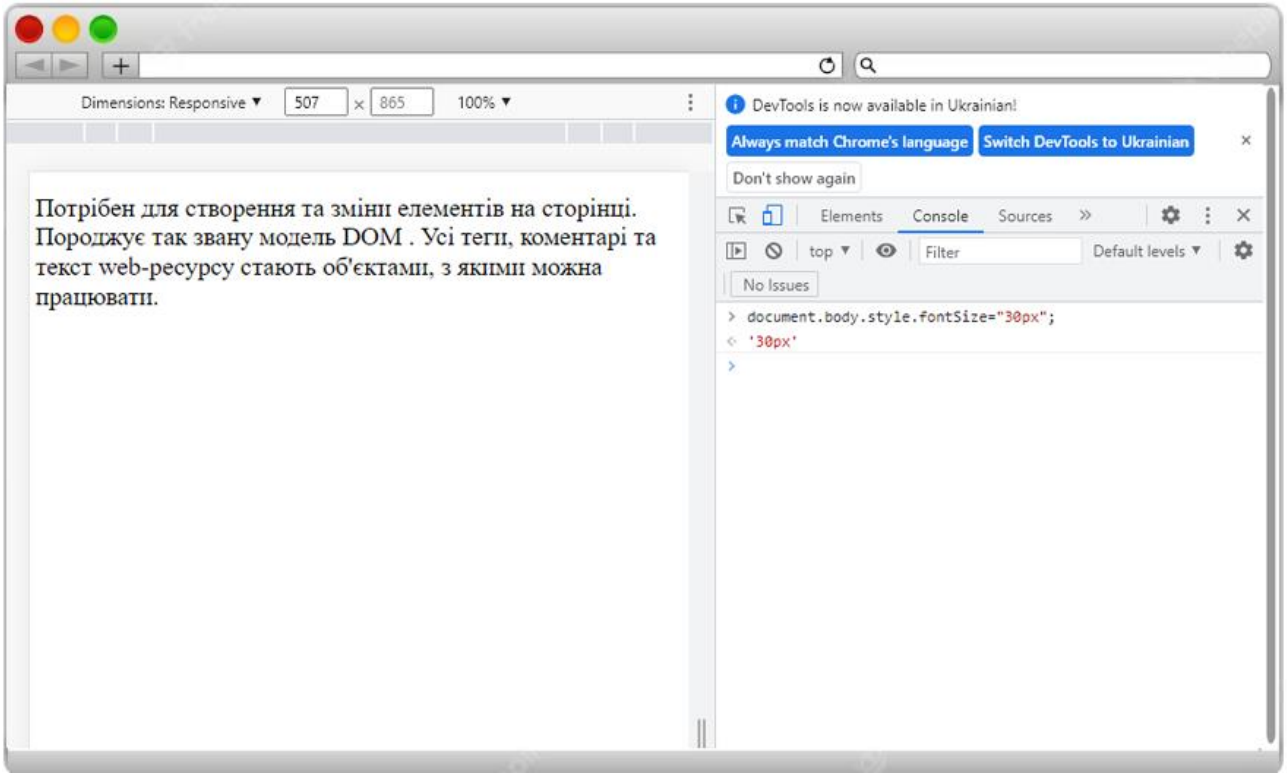


Рис. Зміна розміру шрифту на сторінці

Об'єктна модель браузера – BOM

Для доступу до інструментів самого браузера використовується BOM . З цього оточення можна отримати дані про переглядача, операційну систему, поточну адресу ресурсу, історію відвідування сторінок та ін.

Приклад - Консоль браузера

```
// Отримуємо адресу поточної сторінки
> location.hostname
< www.google.com.ua
// Отримуємо тип операційної системи
> navigator.platform
< "Win32"
// Отримуємо висоту екрану
> screen.height
< 865
```

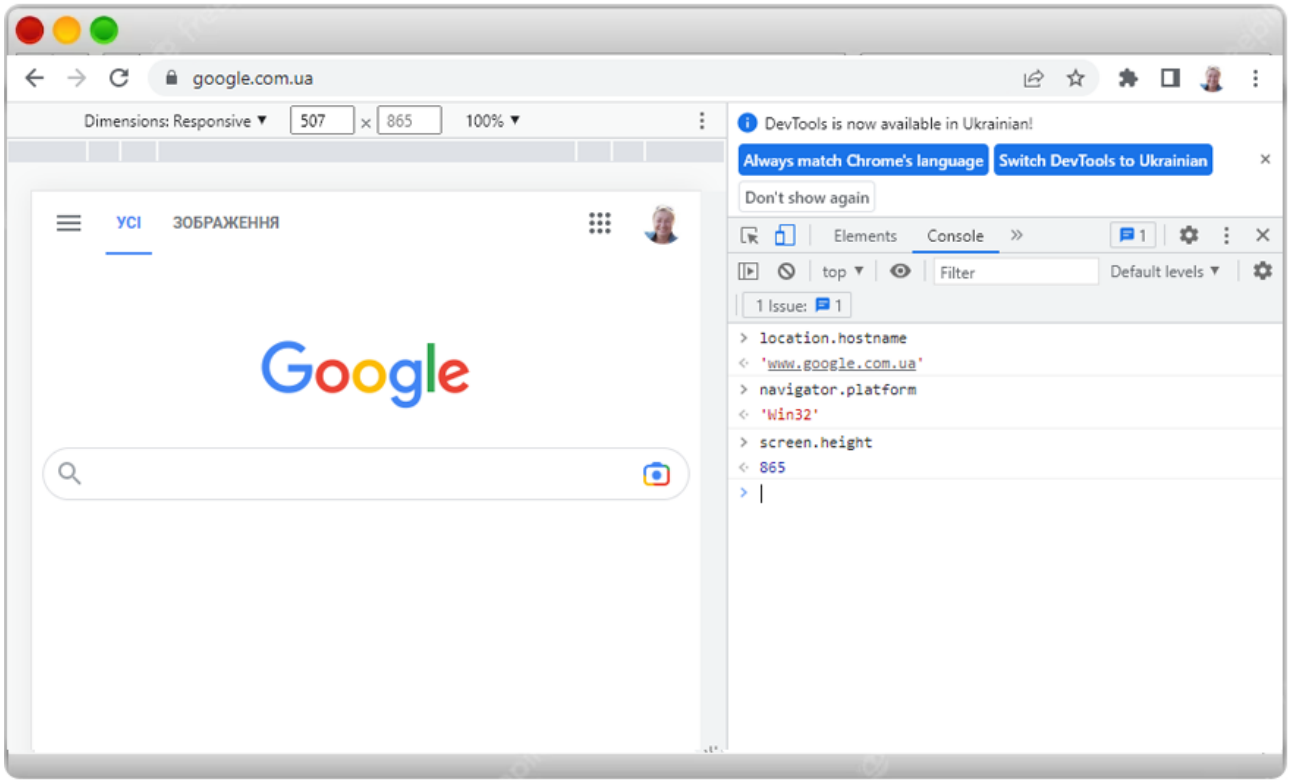


Рис. Робота з об'єктами BOM

Підключення скриптів до сайту

Впроваджувати JavaScript на сторінку можна кількома способами :

- Безпосередньо в HTML-код.
- З зовнішніх файлів (у заголовній частині документа або перед тегом <body>).

Найзручніше створювати зовнішні скрипти, які на кшталт таблиць стилів, що підключаються, прикріплюються до документа. Щоб js-код спрацював після повного завантаження всіх елементів сторінки, його прийнято розміщувати перед тегом <body>. Однак, ніхто не забороняє давати посилання на нього та в заголовку з використанням ключового слова defer.

Приклад - HTML

```
<!DOCTYPE html>
<html lang="uk">
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<title>Практика JS</title>
	<script src="learning.js" defer></script>
</head>
<body>
	<h1>Вивчаємо JS</h1>
</body>
</html>
```

Атрибут defer передбачає, що спочатку завантажиться весь вміст ресурсу та його стилі, а потім буде застосований скрипт.

Для того, щоб на консолі не виводилася помилка про відсутність файлу `learning.js`, створимо файл learning.js, який прикріпимо до сторінки..

Навігація по DOM

Відомо, що HTML-сторінка складається із тегів. Модель DOM сприймає їх як об'єкти (і не лише їх). За допомогою JavaScript можна отримати доступ до будь-яких об'єктів, переглянути їх вміст, змінити тощо. Веб-документ у разі матиме деревоподібну структуру, у якій виділяють батьків, нащадків, сусідів. Наведена вище HTML-сторінка має наступну DOM-структуру:

DOM-структура

HTML

```
├ HEAD
├─ #text
├─ META
├─ #text
├─ META
├─ #text
├─ TITLE
├── #text Практика JS
├─ #text
├─ SCRIPT
└── #text
├ BODY
├─ #text
├─ H1
└── #text Вивчаємо JS
```

Крім безпосередньо самих тегів, як об'єкти виступають текстові дані та коментарі. Навіть перенесення рядків (коли відокремлюємо блоки один від одного) відображатимуться у вигляді текстового об'єкта.

Для навігації по DOM-дереву слід знати про такі поняття:

1. Кореневі елементи:

    а) document.documentElement. Найвищий вузол, відповідає тегу <html>, містить весь вміст документа.

    б) document.head. Заголовна частина веб-сторінки.

    в) document.body. Тіло документа, вміст тега <body>.

2. Вузол-батько. parentNode – безпосередній нащадок конкретного об'єкта.

3. Діти, нащадки:

    а) childNodes. Колекція нащадків певного об'єкта.

б) firstChild. Перший нащадок.

в) lastChild. Останній дочірній елемент.

4. Сусіди (один рівень ієрархії):

а) nextSibling. Наступний сусід того ж батька.

б) previousSibling. Попередній елемент всередині батька, що знаходиться на тому ж рівні, що і початковий.

Відкриємо представлену вище HTML-сторінку у браузері та у консолі попрацюємо з вивченим матеріалом.

Приклад - Консоль браузера

```
// Виводимо вміст усієї сторінки
> document.documentElement
<html lang="uk">
      <head>…</head>
      <body>…</body>
</html>
// Виводимо нащадків вузла head
> document.head.childNodes
< NodeList(9) [text, meta, text, meta, text, title, text, script, text]
// Отримуємо першу дочірню ноду вузла head та виводимо її першого сусіда
> document.head.childNodes[0].nextSibling
< <meta charset="UTF-8">
```

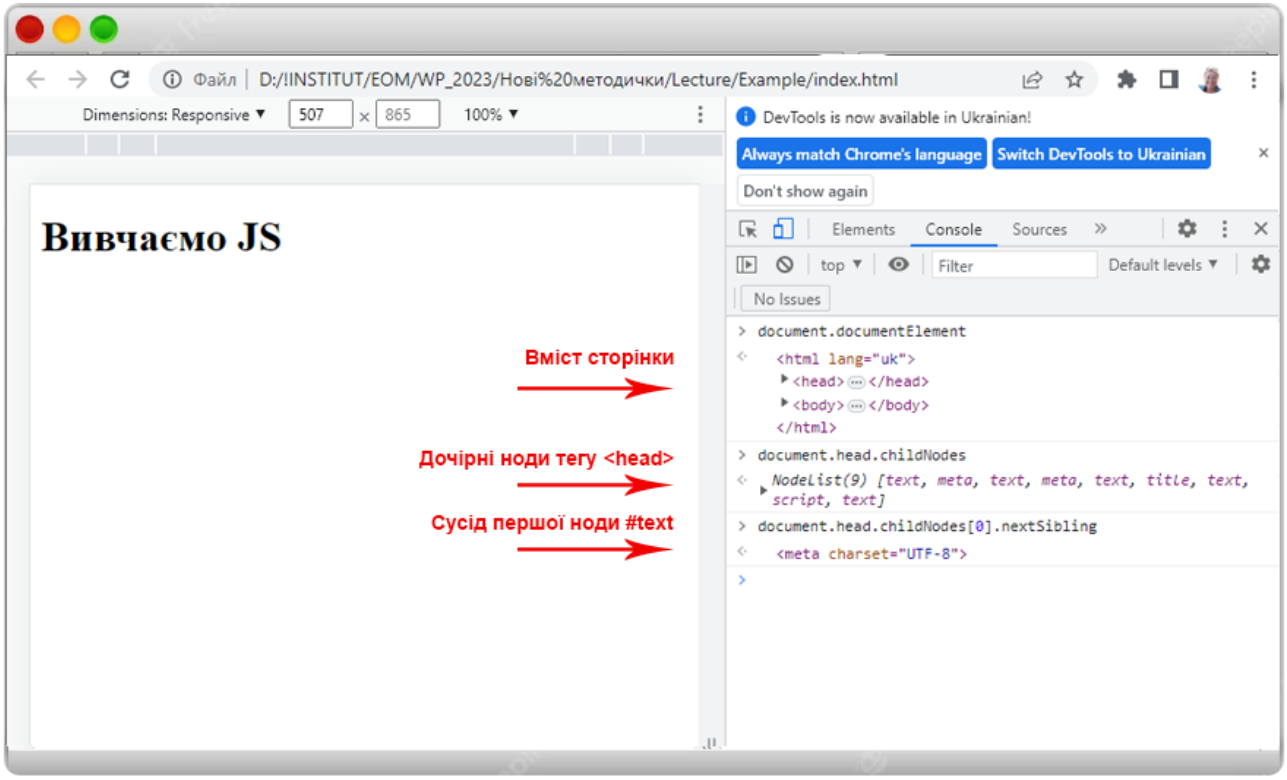


Рис. Робота з нодами DOM-дерева

Перелік вузлів зовні нагадуємо масив (список), але має свої особливості:

- Не змінюється (не можна додати або видалити з нього об'єкт).
- Не підтримує методи масивів.
- Перебирається циклом for…of.
- Є "живим" (видалення або додавання на сторінку елементів автоматично відобразиться на розмірі колекції).

Для роботи з JavaScript створимо файл learning.js, який прикріпимо до сторінки. Тепер у консолі ми отримаємо перелік усіх дев'яти вузлів, що знаходяться всередині тега <head> .

Приклад – JavaScript (тут і далі – файл learning.js)

```
for (let node of document.head.childNodes) {
     console.log(node);
}
```

Результат виконання

```
#text
  <meta charset="UTF-8">
#text
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
#text
  <title>Практика JS</title>
#text
  <script src="learning.js" defer></script>
#text
```

У більшості випадків не надто потрібні додаткові текстові об'єкти DOM-дерева, а потрібен лише список тегів. Для цього є відповідні інструменти (властивості):

- Властивість children – список нащадків HTML-елемента, які самі є тегами.
- Властивість firstElementChild - перший нащадок.
- Властивість lastElementChild – останній нащадок
- Властивість previousElementSibling – попередній сусід елемента.
- Властивість nextElementSibling – наступний сусід елемента.
- Властивість parentElement – батьківський тег.

Приклад - Консоль браузера

```
// Отримуємо нащадків вузла body
> document.body.children
< HTMLCollection(1) [h1]
// Отримуємо батька вузла body
> document.body.parentElement
<html>…</html>
// Отримуємо верхнього сусіда вузла body
> document.body.previousElementSibling
<head>…</head>
```

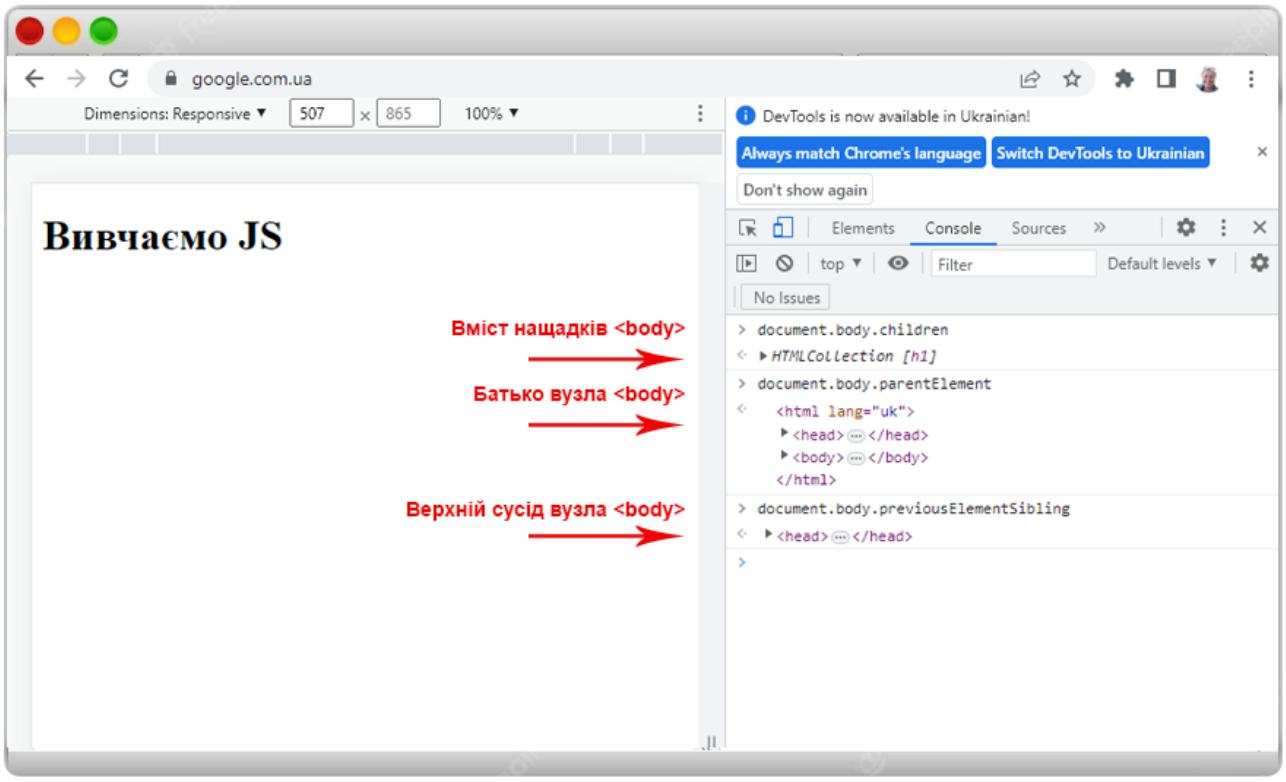


Рис. Робота з тегами всередині структури DOM

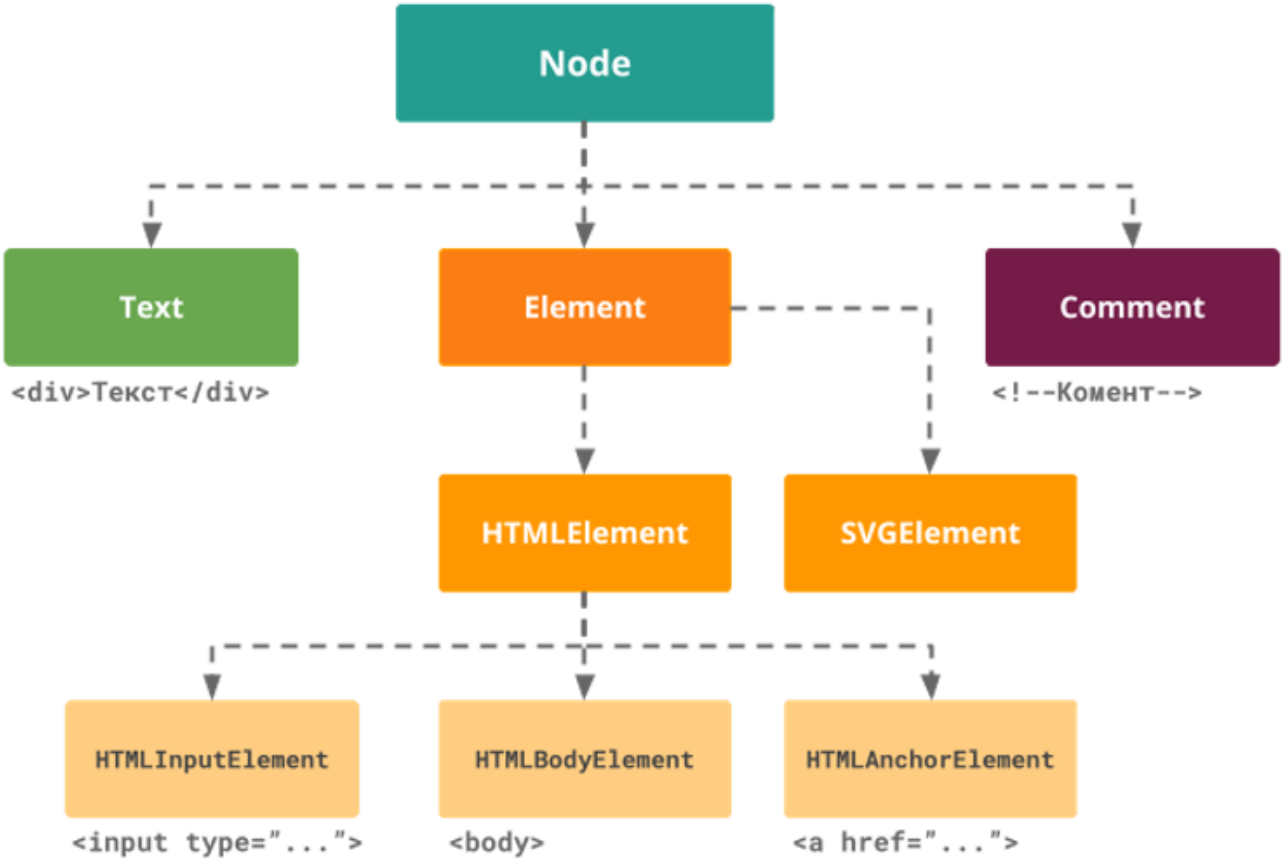
Якщо при виклику властивості childNodes отримуємо колекцію вузлів, children показує HTML-колекцію.

Властивості DOM-вузлів

Вузли в залежності від класу можуть мати власний перелік властивостей та методів. У будь-якому випадку, вони є об'єктами JavaScript, що дозволяє задавати їм атрибути і методи.

DOM-вузли належать до певних класів , що мають ієрархію:

- Клас EventTarget - клас-основа, що дозволяє об'єктам підтримувати події.
- Клас Node - ключовий клас для вузлів, що дозволяє їм бути обробленими за допомогою властивостей та методів: childNodes, nextSibling та інші.
- Клас Element – відповідає за навігацію на рівні елементів та забезпечує їх методами пошуку. Є базою для SVGElement, XMLElement і HTMLElement.
- HTMLElement – дозволяє успадковуватись іншим тегам (HTMLInputElement, HTMLAnchorElement, HTMLBodyElement) і наділяє їх властивостями (click(), focus() та ін).



Щоб зрозуміти належність об'єкта до класу є спеціальна інструкція: instanceof. Дізнатися про приналежність можна і через console.dir().

Приклад - Консоль браузера

```
> document.head instanceof HTMLElement
<true
> document.head instanceof Node
<true
> console.dir(document.body)
< ▶body
```

Раніше застосовувався ще один спосіб: властивість nodeType , але сьогодні він не такий інформативний.

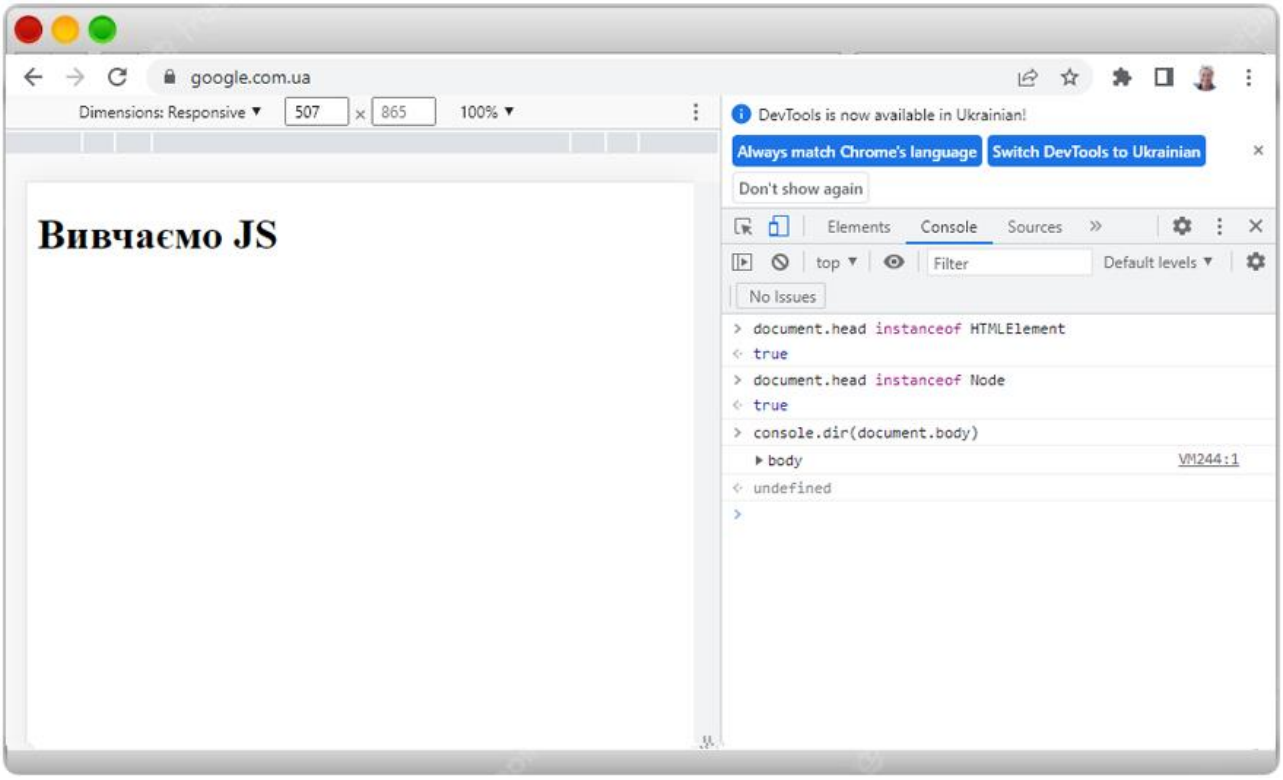


Рис. Властивість instanceof

Приклад - Консоль браузера

```
> document.head.nodeType
< 1
```

Число 1 свідчить, що маємо справу з вузлом-елементом, а число 3 – текстовим.

На основі вузла DOM можна з'ясувати ім'я елемента тегу: nodeName і tagName. Друга властивість є лише для об'єктів класу Element, а перше доступне для всіх.

Приклад - Консоль браузера

```
> document.TagName
< undefined
> document.nodeName
< # document
```

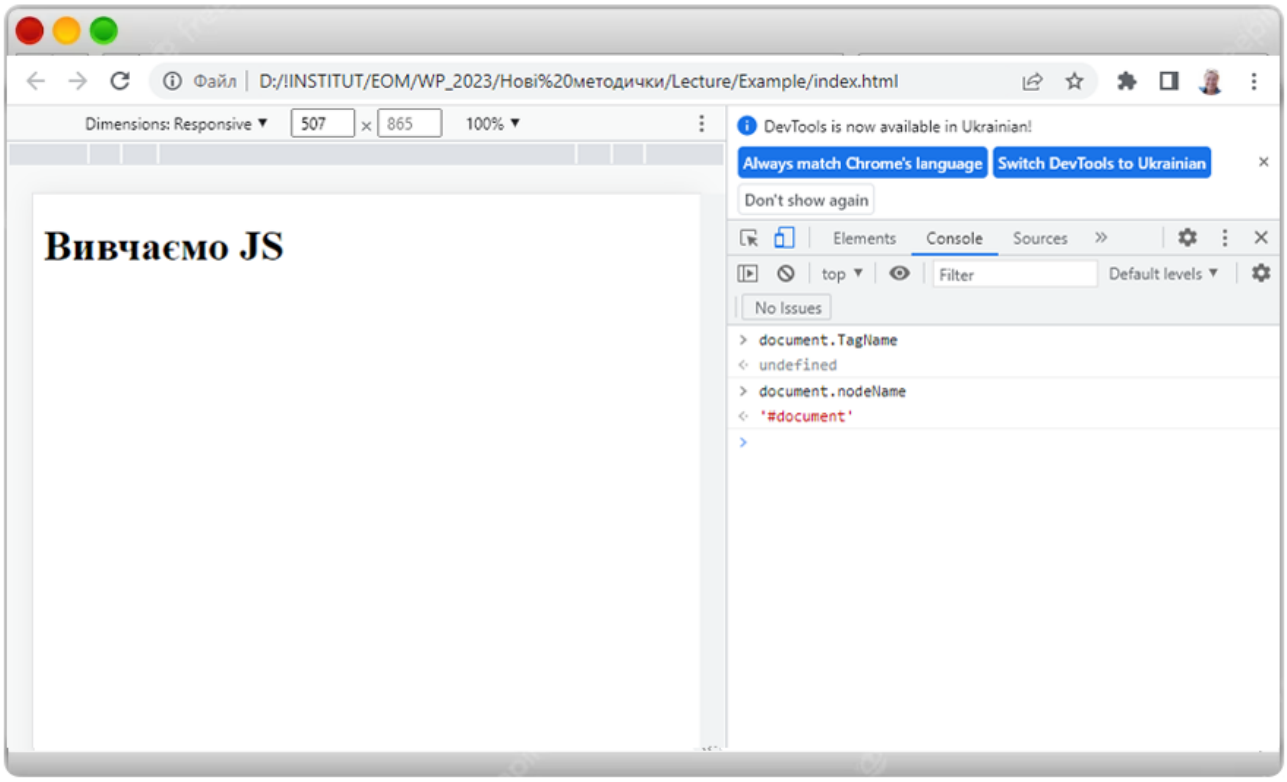


Рис. Виведення імені елемента

Розглянемо деякі інші властивості , що часто використовуються на практиці.

Властивість	Опис
`innerHTML`	Доступ до внутрішнього контенту елемента (представлений у вигляді рядка)
`outerHTML`	Показує вміст елемента разом із ним самим
`nodeValue`	Вміст текстового вузла чи коментаря
`data`	Аналогічно до nodeValue за деякими нюансами (направлено виключно на текст)
`textContent`	Внутрішній текст об'єкта без врахування будь-яких тегів
`hidden`	Відповідає за видимість тегу на сторінці.

Дані властивості можна як отримувати, а й задавати.

Приклад - Консоль браузера

```
//Вміст тіла документа
> document.body.innerHTML
```

```
<h1>Основні властивості вузлів</h1> …
// Вміст тега <body> разом із ним самим
> document.body.outerHTML
< "<body> ↲ <h1>Осн…
// Внутрішній текст всіх вхідних елементів
> document.body.textContent
< Основні властивості вузлів
// Визначаємо, чи схований тег <h1>
> document.body.firstElementChild.hidden
< false
// Ховаємо тег <h1>
> document.body.firstElementChild.hidden = true
< true
// Отримуємо вміст текстового вузла (тут – це перенесення рядка)
> document.body.firstChild.data
< ↲
// Змінюємо перенесення рядка на довільний текст
> document.body.firstChild.data = 'Текст'
< "Текст"
```

Є й інші властивості, характерні як окремих тегів, і загальні для всіх:

Властивість	Опис
`value`	Отримання значення у елементів input, textarea, select
`href`	Посилання
`Id`	Отримання ідентифікатора у разі його наявності
`className`	Назва всіх класів об'єкту

Властивості DOM-об'єктів не слід плутати з атрибутами HTML-елементів. Важливо запам'ятати, що властивості здатні набувати будь-яких значень, а атрибутами можуть бути тільки рядки.

Вбудовані властивості вже були розглянуті, але можна додавати власні, в тому числі і способи.

Приклад - Консоль браузера

```
// Створюємо нову властивість
> document.body.myProperty = 'Створена властивість'
> document.body.myProperty
```

```
< "Створена властивість"
```

Взаємодія з нестандартними, атрибутами користувача здійснюється за допомогою чотирьох методів :

Метод	Опис
`hasAttribute()`	Перевірка наявності атрибута елемента
`getAttribute()`	Отримання значення атрибуту
`setAttribute()`	Створення атрибуту з деяким значенням
`removeAttribute()`	Видалення атрибуту.

Приклад - Консоль браузера

```
// Перевіряємо наявність атрибуту
> document.body.hasAttribute('layout')
< false
// Задаємо новий атрибут
> document.body.setAttribute('layout','flexible')
// Отримуємо створений атрибут
> document.body.getAttribute('layout')
< "flexible"
// Видаляємо атрибут
> document.body.removeAttribute('layout')
> document .body.hasAttribute('layout')
< false
```

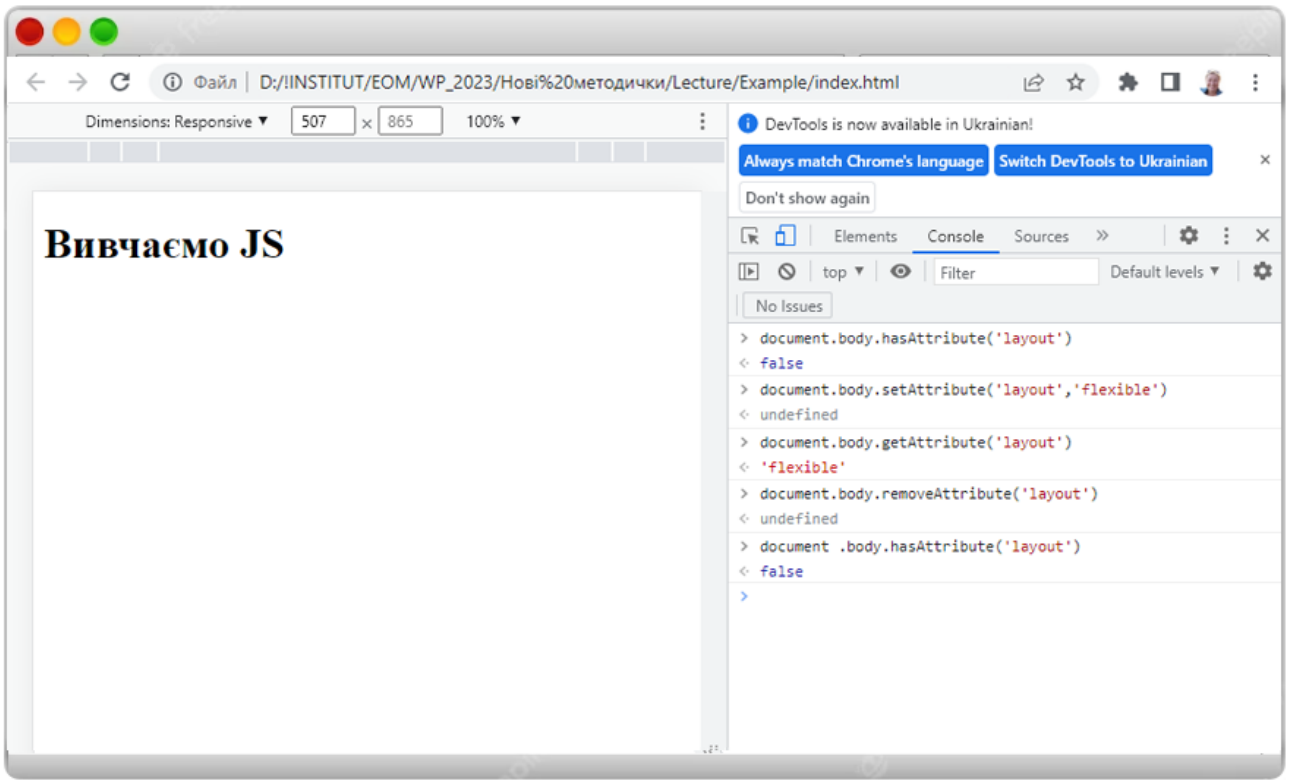


Рис. Вбудовані властивості

Спочатку у тега <body> не було властивості layout. Ми її створили, надали значення, а потім видалили. За допомогою продемонстрованих вище методів, звичайно, можна звертатися і до стандартних атрибутів DOM-об'єктів .

Data-атрибути

Багато хто з вас міг бачити на ряді сайтів деякі дивні властивості, що починаються з data- . Виявляється, вони зручні та розроблені спеціально для програмістів. До них можна отримати доступ через точкову нотацію.

Приклад - HTML

```
<body data-site="https://google.com.ua/" data-name="Google"></body>
```

Приклад - Консоль браузера

```
document.body.dataset.site
'https://google.com.ua/'
document.body.dataset.name
"Yandex"
```

Як видно, щоб звернутися до таких атрибутів потрібно вказати властивість dataset, а потім все те, що йде після data- .

Пошук по дереву документа

Усі розглянуті раніше методи доступу до тегам ґрунтувалися або з їхньої базовості (їм зарезервовані спеціальні імена), або з допомогою родинності-сусідства. Це не завжди зручно, оскільки може породжувати довгі ланцюжки, а також не нести жодної інформативності. Саме для цього в браузерному оточенні є інші способи пошуку елементів:

Функція	Опис
`getElementById()`	Шукає об'єкт за ідентифікатором
`getElementsByName()`	Дозволяє виділити елементи із заданим атрибутом «name»
`getElementsByTagName()`	Виявлення елементів за назвою тегу
`getElementsByClassName()`	Пошук на основі класу
`querySelector()`	Знаходить перший елемент на основі CSS-селектора
`querySelectorAll()`	Шукає всі об'єкти за заданим селектором

Для практичного прикладу створимо наступний набір тегів всередині <body> .

Приклад - HTML

```
<h1 id="first_header">Пошук у дереві документа</h1>
<article class="main_article">Перша стаття</article>
<article class="main_article">Друга стаття</article>
<article class="main_article" name="last">Остання стаття</article>
<h1>Основи DOM</h1>
```

Приклад - Консоль браузера

```
// Виберемо заголовок і отримаємо весь його вміст
> document.getElementById('first_header')
< <h1 id = "first_header">Пошук у дереві документа</h1>
// Виберемо останню статтю виходячи з властивості «name»
> document.getElementsByName('last')
< NodeList[article.main_article]
// Знайдемо всі статті на сторінці за тегом
> document.getElementsByTagName('article')
< HTMLCollection(3) [article.main_article, article.main_article,
article.main_article, last: article.main_article]
// Виведемо список статей із класом «main_article»
> document.getElementsByClassName('main_article')
< HTMLCollection(3) [article.main_article, article.main_article,
article.main_article, last: article.main_article]
// Отримаємо доступ до першої статті
```

```
> document.querySelector('.main_article')
< <article class="main_article"> Перша стаття</article>
// Представимо список усіх заголовків першого рівня
> document.querySelectorAll('.main_article')
< NodeList(3) [article.main_article, article.main_article,
article.main_article]
```

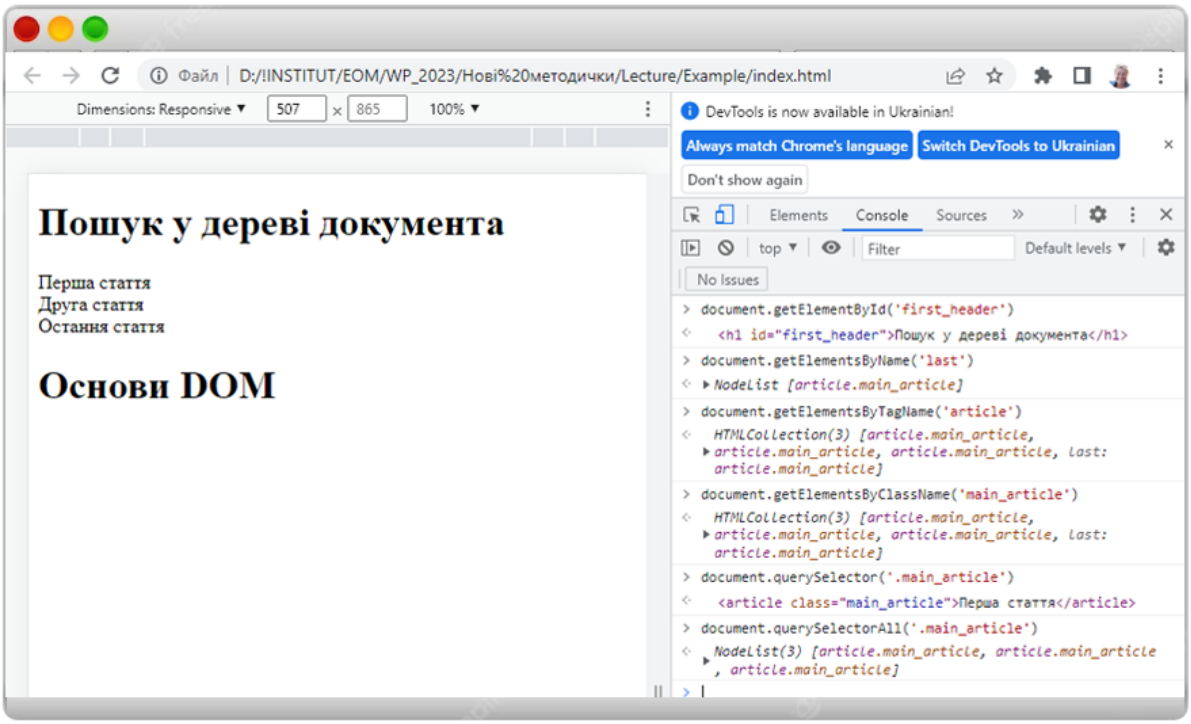


Рис. Способи пошуку елементів

Практично у всіх випадках, що передбачають множинний пошук сторінок, отримуємо живу колекцію об'єктів. Є один виняток: функція querySelectorAll(). Функції querySelectorAll() і querySelector() дозволяють застосовувати CSS-селектори будь-якої складності.

Внесення змін до DOM-дерева

За допомогою JavaScript можна не лише знаходити елементи в веб-документі або змінювати їх властивості, але й вносити зміни до структури DOM-дерева:

- Створювати нові теги.
- Видаляти наявні.
- Замінювати їх на інші.

Для цього застосовуються такі інструменти:

Функція	Опис
`createElement()`	Створення певного тегу
`createTextNode()`	Оголошення текстового вузла з деяким вмістом

Функція	Опис
`remove()`	Видалення елемента
`cloneNode()`	Повна копія бажаного об'єкту
`append()`	Додавання вузла в кінець
`prepend()`	Вставка об'єкта на початок вузла
`before()`	Додавання елемента перед вузлом
`after()`	Вставлення об'єкта після вузла
`replaceWith()`	Заміна вмісту

Для практики будемо користуватися наступним HTML-шаблоном.

Приклад - HTML

```
<article class="main-article">Основна стаття</article>
<p>Деякі абзаци</p>
<section>Додаткова секція</section>
```

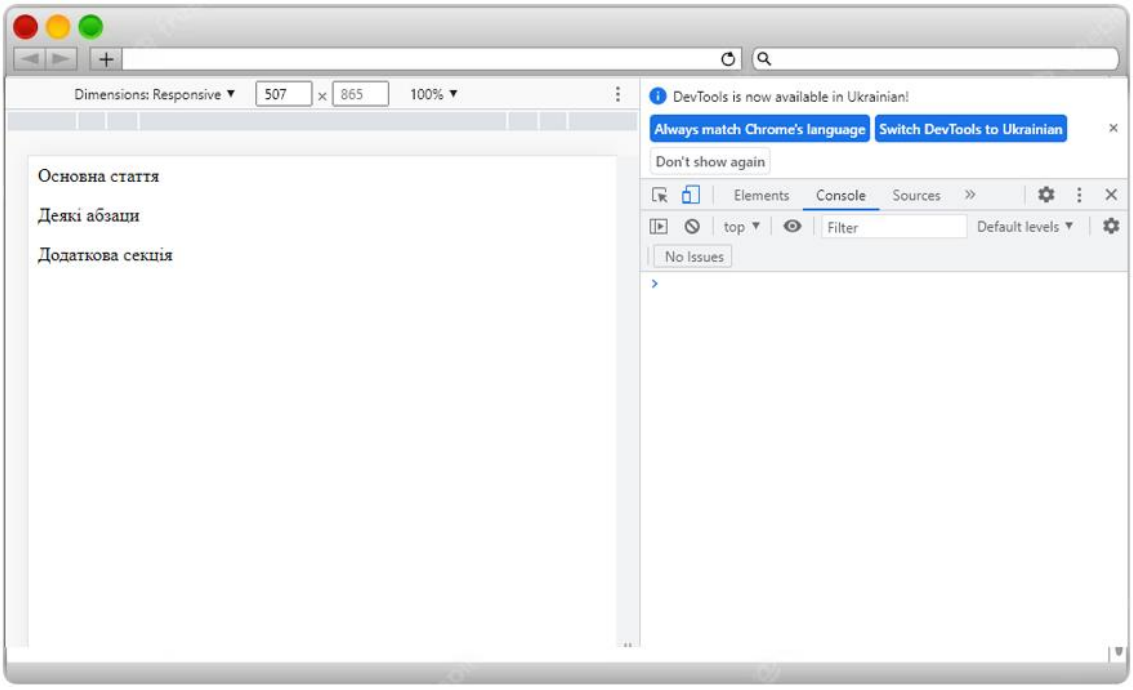


Рис. Сторінка до змін

Створимо текстовий елемент та заголовок першого рівня на початку сторінки

JavaScript - Файл learning.js

```
const text = document.createTextNode('Деякий текст');
const h1 = document.createElement('h1');
document.body.prepend(text);
document.body.prepend(h1);
h1.append('Заголовок сторінки')
```

Ми сформували текстовий об'єкт та тег <h1>, які розмістили на самому початку сторінки. Всередину заголовка додатково додано зміст.

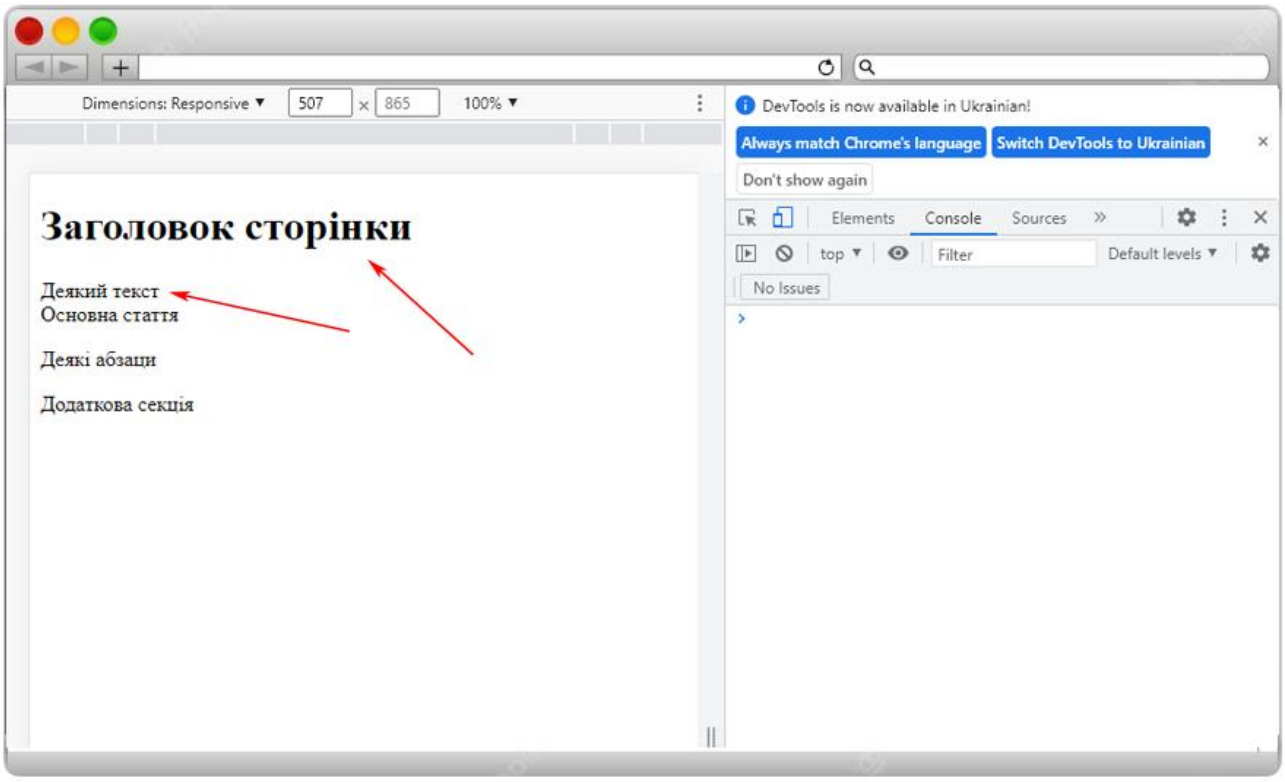


Рис. Сторінка після додавання тексту та заголовка

Копіювання абзацу та додавання його після секції із заміною тексту

JavaScript - Файл learning.js

```
const p = document.querySelector('p');
const pModified = p.cloneNode(true);
pModified.innerHTML = '<strong>Новий абзац</strong>'
const section = document.getElementsByTagName('section')[0];
section.after(pModified)
```

Проведені маніпуляції дозволили скопіювати абзац, замінити його вміст та вставити до кінця документа.

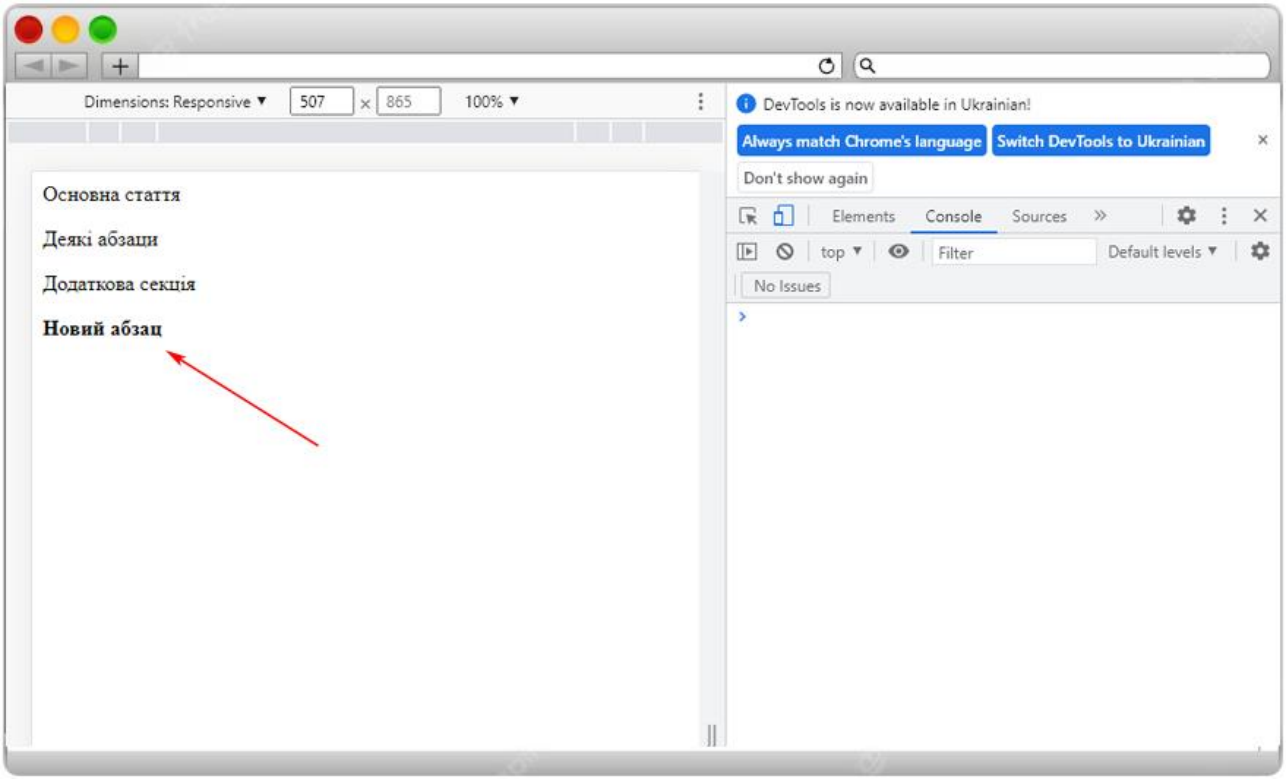


Рис. Сторінка після копіювання та модифікації абзацу

Заміна всіх тегів на сторінці заголовками другого рівня

JavaScript - Файл learning.js

```
const tags = document .body.children
for (let tag of tags) {
      const h2 = document.createElement('h2')
      h2.innerText = 'Залишаться лише заголовки!'
      tag.replaceWith(h2)
}
```

Таким чином зроблено заміну абзацу, статті та секції на заголовки <h2> з текстом Залишаться тільки заголовки! .

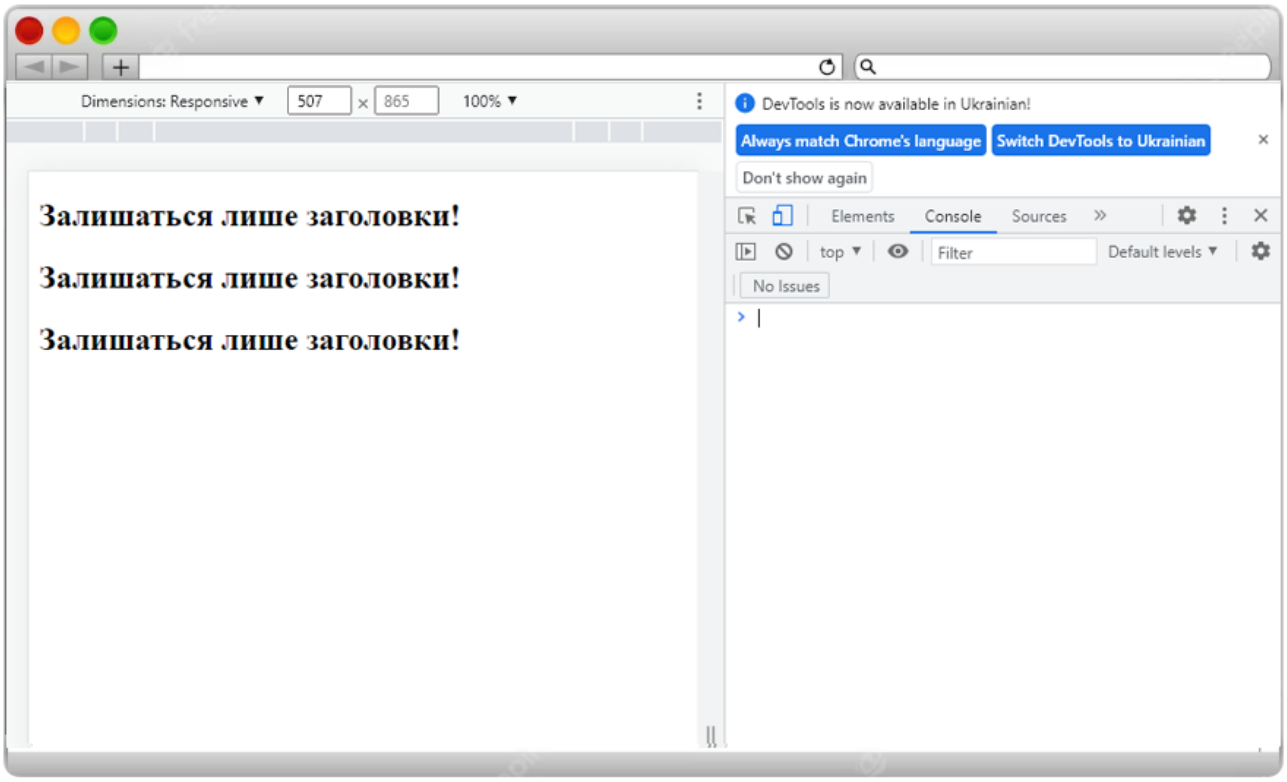


Рис. Заміна всіх елементів сторінки заголовками

Додавання тексту із спеціальними символами

Наведені вище способи додають текст лише безпечним способом (усі спеціальні символи перетворюються. Наприклад, знак > переводиться в &gt). Щоб отримати можливість впроваджувати в документ рядки з тегами, застосовується метод insertAdjacentHTML(). Додамо до та після абзацу посилання з виділеним жирним шрифтом.

JavaScript - Файл learning.js

```
const p = document.querySelector('p');
p.insertAdjacentHTML('beforebegin',
'<a href="https://www.google.com/"><strong>Google</strong></a>');
p.insertAdjacentHTML( 'afterend',
'<a href="https://bing.com/"><strong>Bing</strong></a>');
```

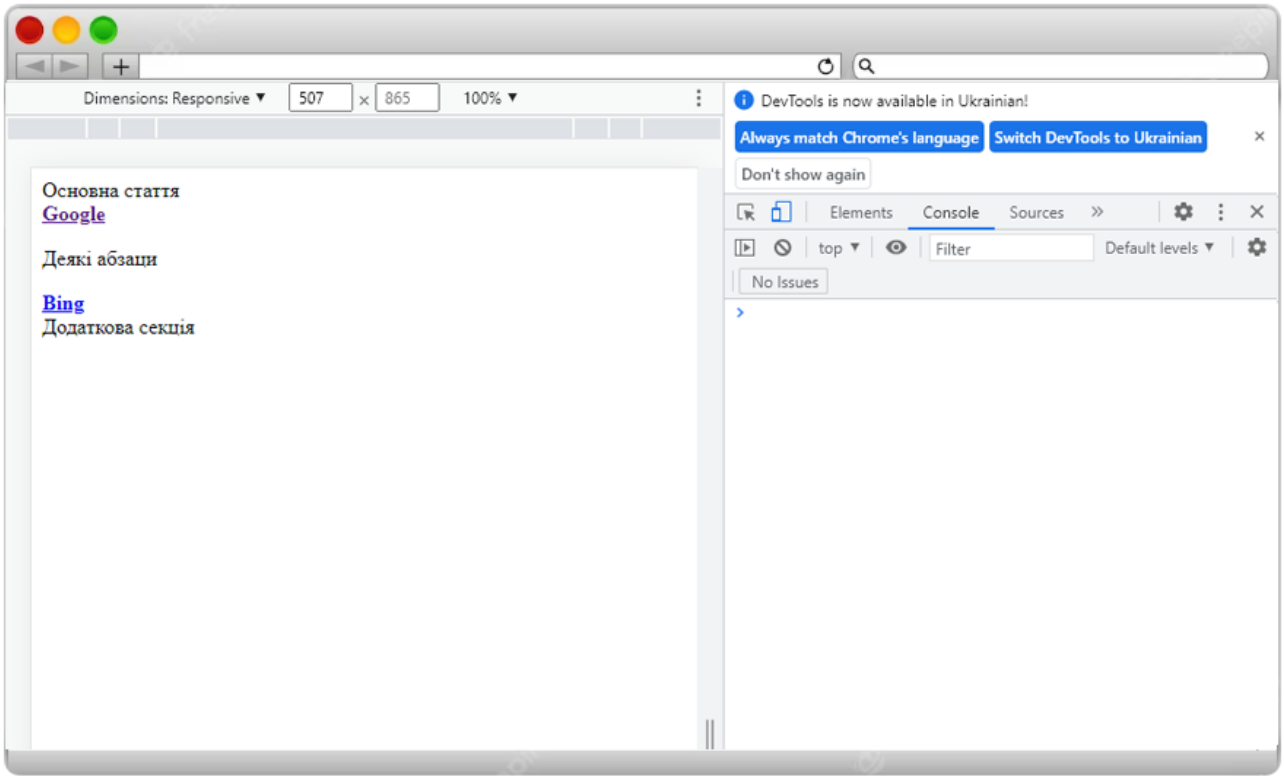


Рис. Додавання тексту за допомогою методу insertAdjacentHTML()

Управління стилями елементів

Взаємодія з класами зміни HTML-елементів може здійснюватися за допомогою властивості style , і безпосередньо зверненням до класів напряму. Переважним вважається другий спосіб, але в деяких ситуаціях неможливо обійтися без першого (наприклад обчислення координат об'єкта).

Весь список доступних CSS-атрибутів застосовується в JavaScript для визначення необхідних значень. Насамперед ознайомимося з методами додавання, видалення, зміни класів, а також перевірки їх наявності. Як шаблон використовуємо наступний HTML-документ.

Приклад - HTML

```
<!DOCTYPE html>
<html lang="uk">
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width, initial-scale=1.0">
	<title>Властивості вузлів</title>
	<script src="learning.js" defer></script>
	<style>
	.colored{
		color : darkorange;
	}
	</style>
</head>
```

```
<body>
      <ul class="colored">
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
      </ul>
      <ul>
      <li>A</li>
      <li>Б</li>
      <li>У</li>
      <li>Г</li>
      <li>Д</li>
      </ul>
</body>
</html>
```

Маємо 2 списки, першому з яких надано клас colored, що робить текст темно-оранжевим.

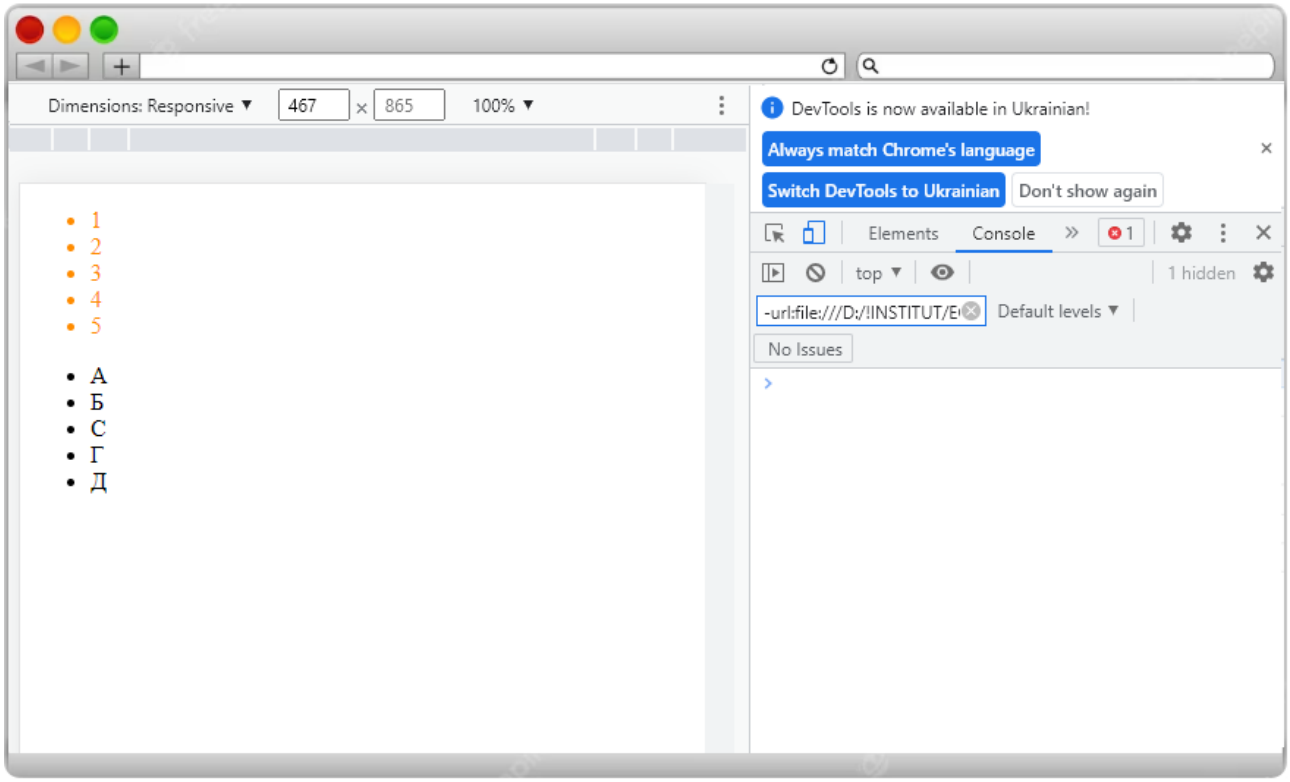


Рис. Зовнішній вигляд шаблону сторінки

Виведення вмісту атрибуту "class"

Приклад - Консоль браузера

```
// Отримуємо елементи з тегом <ul>
> const uls=document.querySelectorAll('ul')
```

```
// Для кожного елемента отримуємо ім'я класу
> for (let tag of uls) {
      console.log(tag.className)
}
< colored
<''
```

У першого списку ми виявили клас colored , а у другого немає жодного класу.

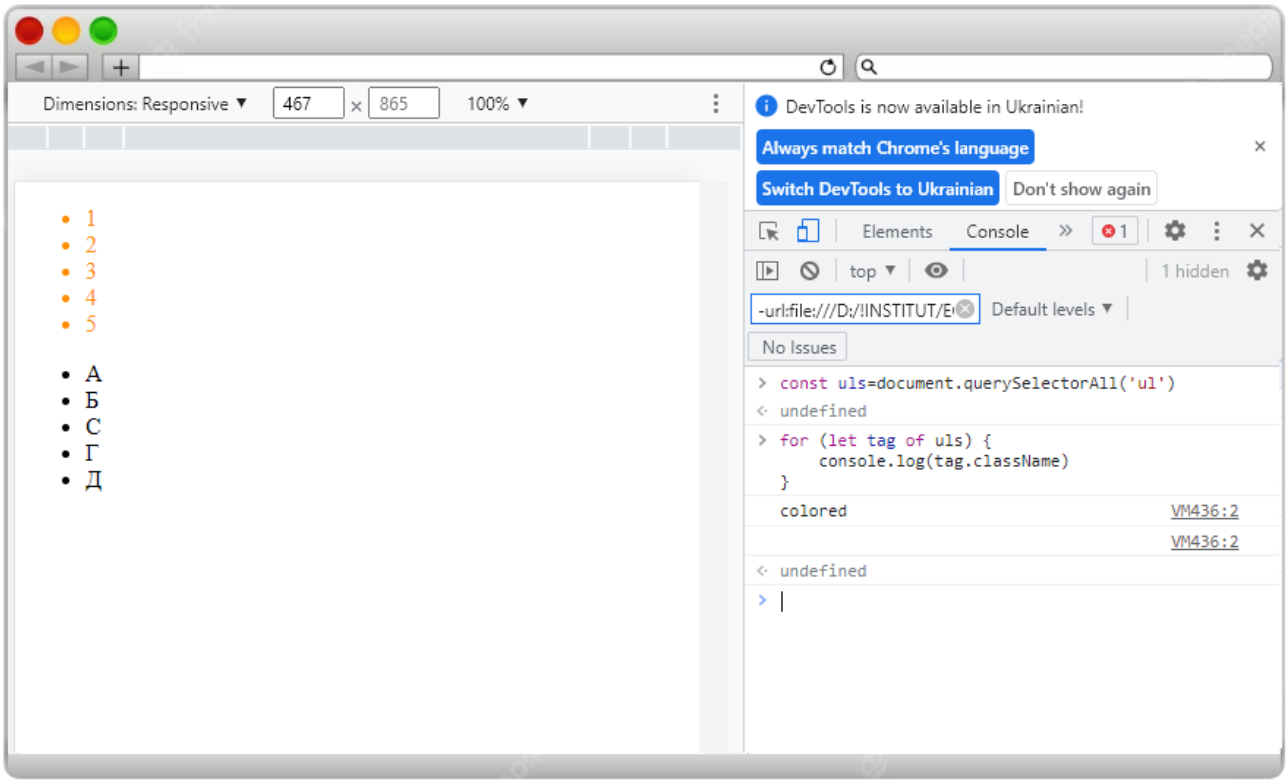


Рис. Виведення вмісту атрибуту "class"

Зміна класів

Об'єкт classList дозволяє змінювати, видаляти чи додавати класи до елементів. Тут є 4 методи:

- Метод add() – додати новий клас.
- Метод remove () - Видалити клас у тегу.
- Метод toggle() – додати клас за відсутності, інакше – видалити.
- Метод contains() – перевіряє наявність класу.

Приклад - Консоль браузера

```
// Отримуємо елементи з тегом <ul>
> const uls = document.querySelectorAll('ul');
> for (let tag of uls) {
      console.log(tag.classList.contains('colored'));
// Видаляємо клас colored для першого списку і навпаки додаємо його для другого
tag.classList.toggle('colored');
```

```
    console.log(tag.classList.contains('colored'));
}
< true
< false
< false
< true
```

Перший елемент <ul> спочатку має шуканий клас, а потім ми його видаляємо. Для другого елемента ситуація обернена.

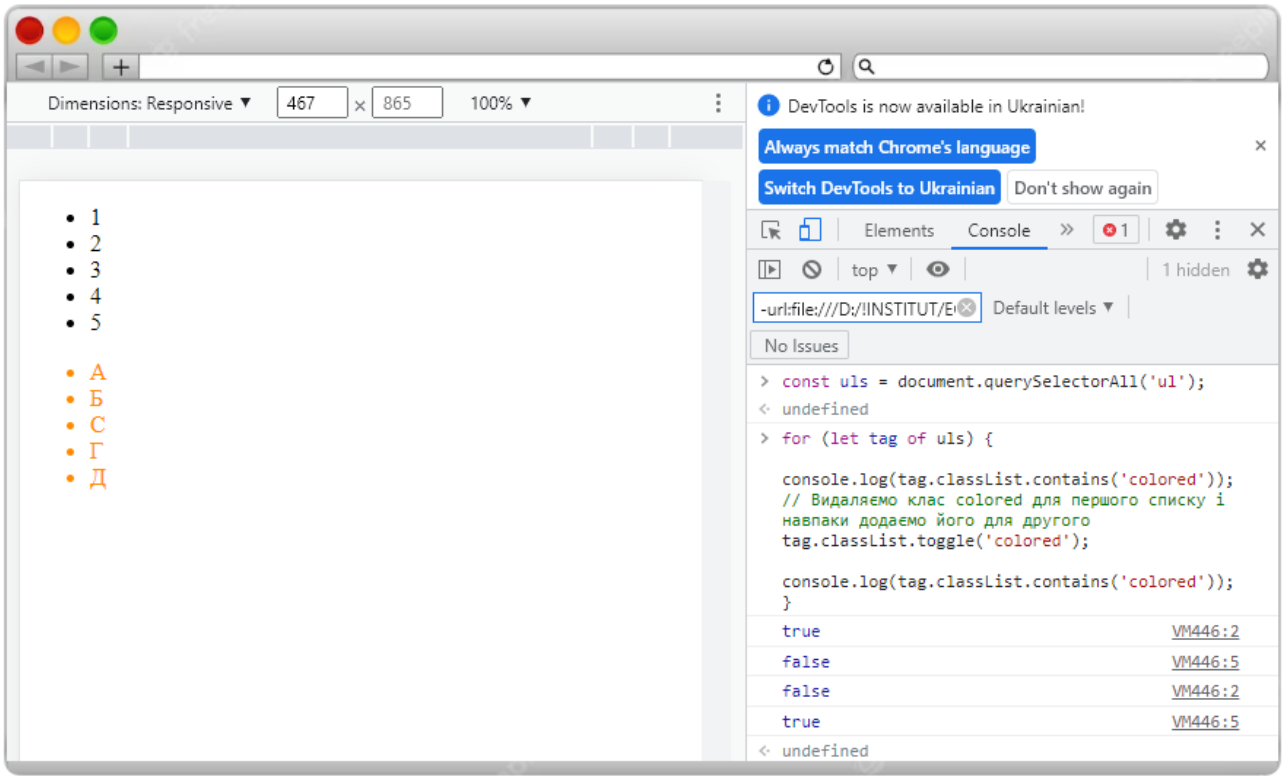


Рис. Зміна зовнішнього вигляду списків після додавання/видалення класу colored

Зміна атрибутів тегів

Змінювати та визначати конкретні атрибути тегів досить просто: потрібно лише до них звернутись за допомогою відповідних методів. Важливо запам'ятати, що в JavaScript не використовуються дефіси, тому складові властивості пишуться у нотації:

CSS властивість -> JS властивість

`font-size(CSS) => fontSize(JS).`

Важливо

Не рекомендується використовувати скорочені найменування атрибутів (наприклад, padding). Краще конкретно вказувати необхідний атрибут (padding-left).

Як ілюстрацію зміни атрибутів попрацюємо зі списками.

Приклад - Консоль браузера

```
// Поміняємо маркери на квадрати
> document.querySelector('.colored').style.listStyleType = 'square'
// Збільшимо шрифт
```

```
> document.querySelector('.colored').style.fontSize = '2rem'
```

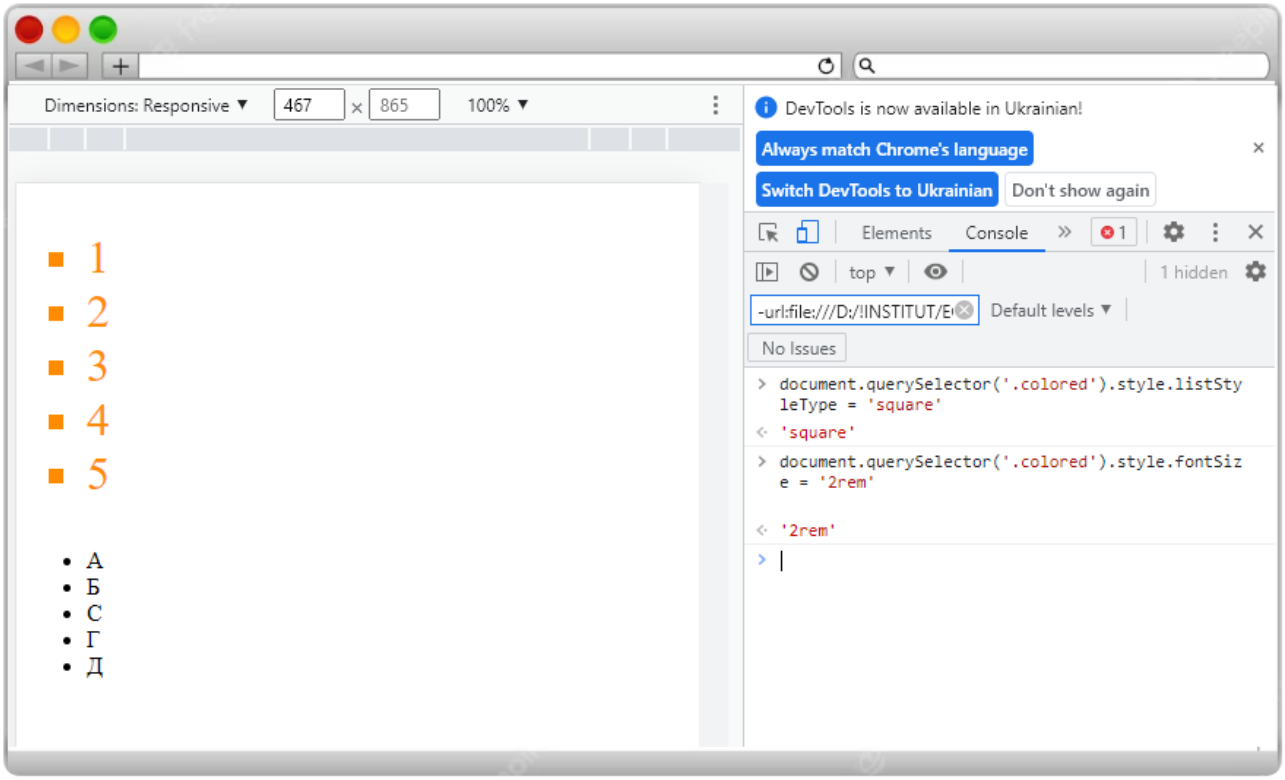


Рис. Сторінка після зміни маркерів та збільшення шрифту

Скидання стилів

Для скидання стилів застосовується властивість cssText. Воно використовується не часто, оскільки повністю переписує стилістику, що може призвести до непередбачених результатів.

Приклад - Консоль браузера

```
> document.querySelector('.colored').style.cssText = `
      list-style-type: "-=- ";
      list-style-position: inside;
      text-align: center;
`
```

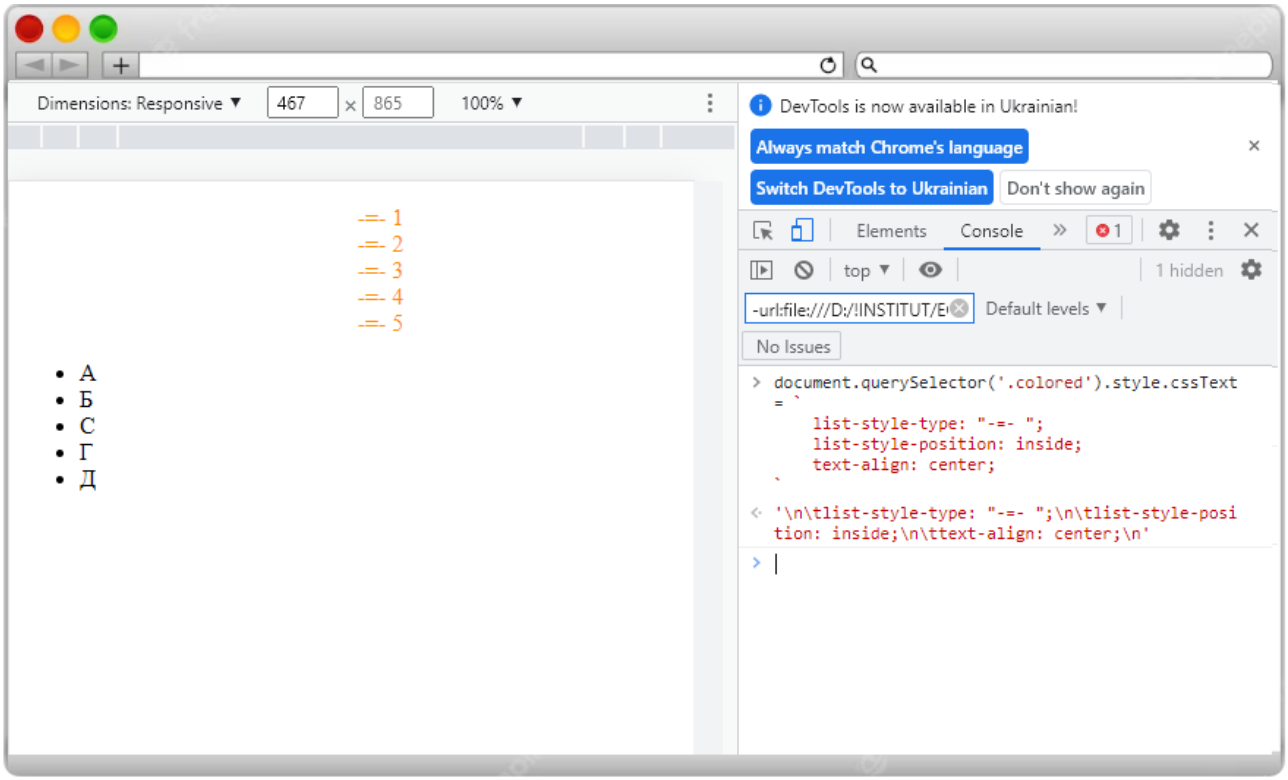


Рис. Сторінка після зміни стилю списку за допомогою cssText

Отримання стилів елементів

Важливе завдання – з'ясувати стилі елемента. Метод getComputedStyle() дозволяє вирішити цю проблему.

Приклад - Консоль браузера

```
> getComputedStyle(document.querySelector('.colored')).display
< block
> getComputedStyle(document.querySelector('.colored')).marginBottom
< 16 px
> getComputedStyle(document .querySelector('.colored')).backgroundColor
< rgba(0,0,0,0)
```

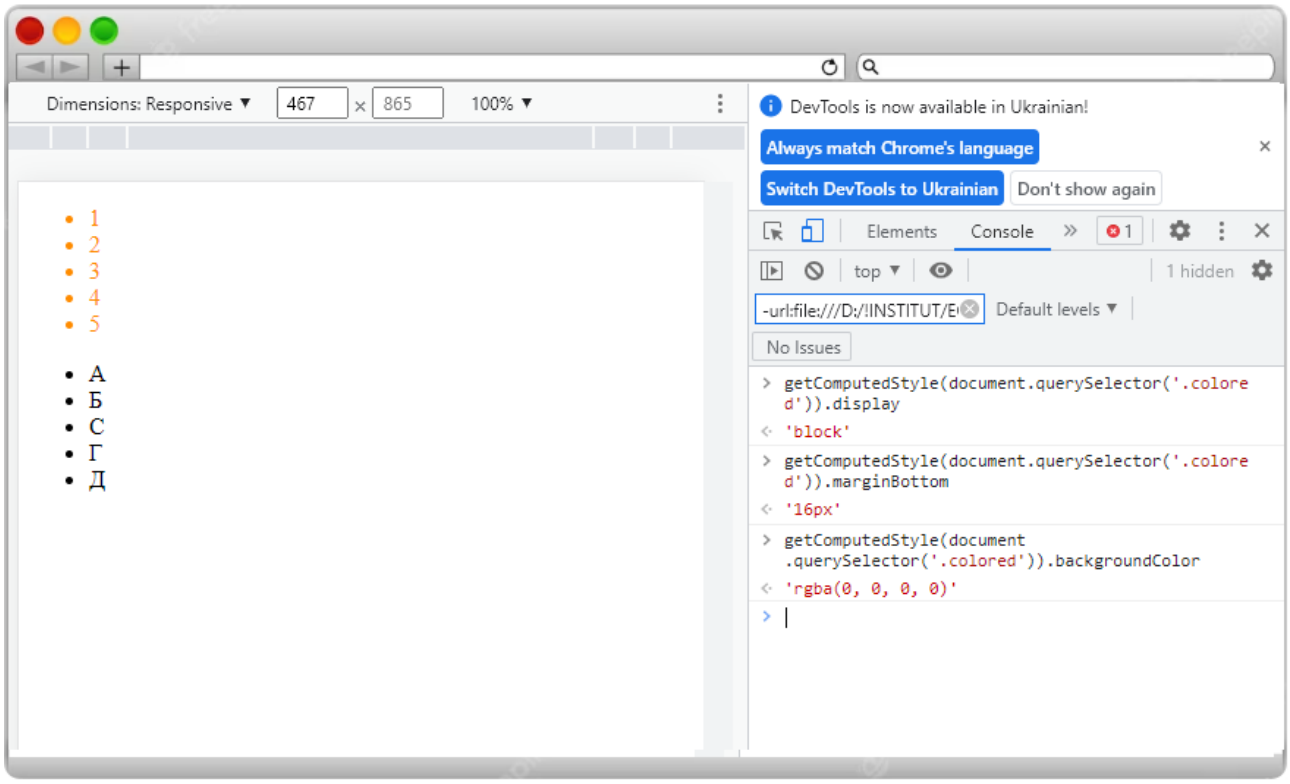


Рис. Отримання стилів елементів

Наведений список властивостей далеко не повний. Для роботи з системою координат, розмірами вікна, прокручуванням є свої інструменти. Ознайомитись з ними можна в офіційній документації (на сайті MDN).

Браузерні події

Браузер постійно повідомляє про різні події (events), і до кожного з них є спосіб визначити деякий обробник. Він є функцією, яка реагує на подію. Наведемо частковий перелік подій:

Подія	Опис
`click`	Найчастіше використовується, спрацьовує при натисканні лівої кнопки миші
`contextmenu`	Реагує на праву кнопку миші
`mousemove`	Пов'язано з рухом мишки
`submit`	Спрацьовує при надсиланні форми
`keydown, keyup`	Реагує на натискання та відпускання кнопки на клавіатурі

Подія	Опис
`transitionend`	Реагує на завершення анімації
`offline`	Спрацьовує за відсутності доступу до мережі

Призначати обробники подій можна кількома способами:

Обробка події через властивість тега

Обробники прописуються в HTML-коді:

Приклад - HTML

```
<body onclick = "console.log('Натиснули!')" ></body>
```

При натисканні в будь-якому місці веб-сторінки в консолі буде виведено повідомлення Натиснути! .

Обробка події через вибір елемента у js-файлі

Даний спосіб є більш зручним та наочним:

Приклад – JavaScript

```
function clicked() {
      console.log('Натиснули!');
}
document.body.onclick = clicked;
```

В даному випадку не потрібно задавати властивість onclick для тега <body> , достатньо лише привласнити йому обробник у скрипті.

Обробка події через спеціальні методи

Сюди належать такі методи:

1. addEventListener()
2. removeEventListener()

Функція addEventListener() дозволяє надавати елементам більше одного обробника подій, а також працює не тільки з тегами, але і будь-якими DOM-об'єктами.

Приклад – JavaScript

```
function one() {
      console.log('Нас тут багато');
}
function two() {
      console.log('Повірте!');
}
function three() {
```

```
      console.log('Дякую методу addEventListener');
}

main = document.documentElement;
main.addEventListener('click', one);
main.addEventListener('click', two);
main.addEventListener('click', three);
```

Одного натискання кнопки миші достатньо, щоб спрацювало 3 обробники подій.

Параметр event функції-обробника

Найчастіше потрібно не просто обробити якусь подію, а й зрозуміти деталі того, що саме сталося. У цьому випадку функції-обробники передається параметр event, що має масу корисних властивостей.

Приклад – JavaScript

```
function getCoords(event) {
      console.log(event.type);
      console.log(event.clientX + 'x' + event.clientY)
}
main = document.documentElement;
main.addEventListener('click', getCoords);
```

За допомогою функції вище ми довідалися про тип події, а також координати миші в момент натискання.

Етапи події (занурення, досягнення мети, спливання)

Коли на сторінці розташована велика кількість об'єктів, то натискання на одному з них (якщо має ряд батьків) викличе спрацювання обробників (якщо вони присутні) у його батьків. Так, якщо зроблено клік по абзацу всередині тега <article>, то він це помітить. Цей принцип називається спливанням.

З іншого боку, ми не завжди вітаємо таку поведінку. Може знадобитися реакція браузера тільки на конкретну подію, а не на весь ланцюжок батьків. Щоб дізнатися, який елемент спричинив початкове спрацювання події, використовують властивість event.target. Навіть якщо подію надано верхньому елементу, то виявиться конкретний дочірній об'єкт, який був задіяний.

Приклад - HTML

```
<div>
      <h4>Спливання</h4>
      <p><span>Певний текст текст</span></p>
</div>
```

Приклад – JavaScript

```
function who(event) {
    console.log(event.target.tagName);
```

```
}
main = document.documentElement;
main.addEventListener('click', who);
```

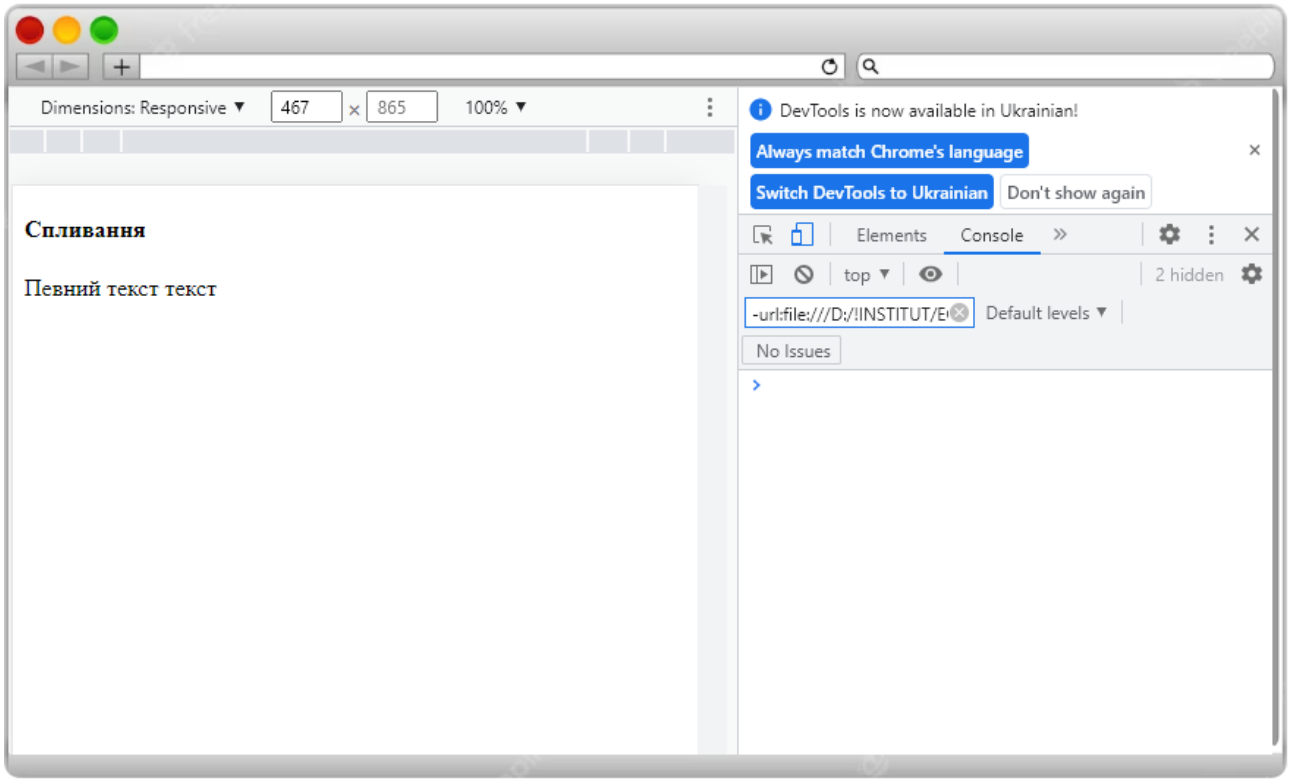


Рис. Властивість event.target

Коли ми будемо кликати по області документа в різних місцях, отримуватимемо імена різних тегів: <h4>, <html>, <p> тощо.

Щоб сплив припинився, застосовується метод stopPropagation() . Важливо усвідомити, що випливання – річ корисна, і припиняти його через незнання – нерозумно.

Говорячи в цілому про цикл події варто відзначити, що він складається з трьох стадій :

1. Занурення (capturing) - від верхнього батька до цільового об'єкта.
2. Досягнення мети (target) – досягається елемент, у якому сталася подія.
3. Спливання (bubbling) - переміщення від дочірнього елемента до батьків.

Занурення використовують рідко, але про нього слід знати. Щоб побачити його, для addEventListener() передають третій параметр.

Приклад – JavaScript

```
for(let tag of document.querySelectorAll('*')) {
    tag.addEventListener("click", event => console.log(`Занурюємося вглиб: ${tag.tagName}`), true);
    tag.addEventListener("click", event => console.log(`Піднімаємося бульбашкою: ${tag.tagName}`));
  }
```

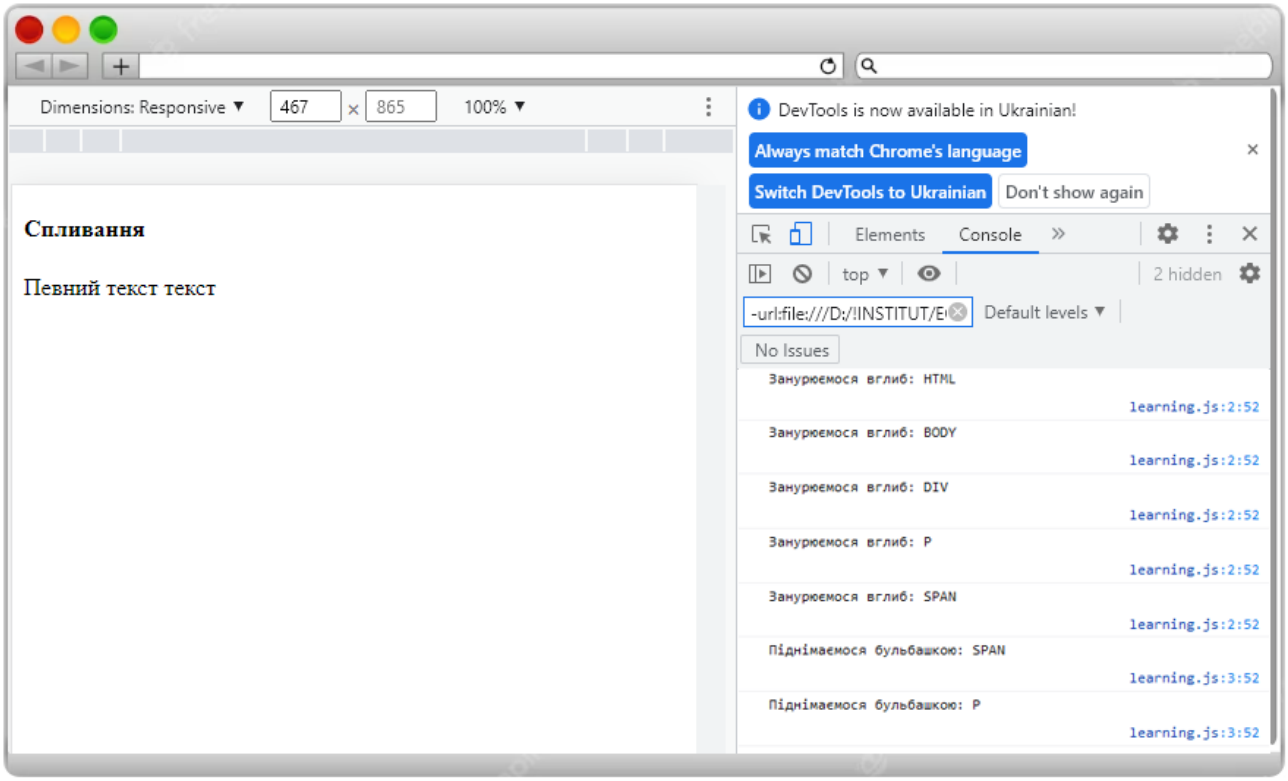


Рис. Властивість addEventListener()

Результат виконання (при натисканні <h4>)

- Поринаємо вглиб: HTML
- Поринаємо вглиб: BODY
- Поринаємо вглиб: DIV
- Поринаємо вглиб: H4
- Піднімаємося бульбашкою: H4
- Піднімаємося бульбашкою: DIV
- Піднімаємося бульбашкою: BODY
- Піднімаємося бульбашкою: HTML

Як видно, процес зачіпає кореневий елемент і доходить до цільового, а потім йде у зворотному порядку.

Делегування подій

На завершення поговоримо про делегування подій . Воно потрібне для того, щоб зменшити кількість коду. Припустимо, у батьківського елемента є велика кількість нащадків. Кожному з них ми б хотіли присвоїти одну й ту саму подію. Найзручніше задати його предку, а залежно від положення миші, наприклад, передавати його лише дочірньому об'єкту.

Приклад - HTML

```
<!DOCTYPE html>
<html lang="ru">
<head>
     <meta charset="UTF-8">
```

```
    <meta    name="viewport"    content="width=device-width,    initial-
scale=1.0">
    <title>Обробник подій событий</title>
    <script src="learning.js" defer></script>
    <style>
      table {
          border: 2px solid blue;
    }
    td {
          background-color: mediumturquoise;
          border: 1px solid black;
          height: 50px;
          width: 200px;
          text-align: center;
    }
     .colored {
          background-color:greenyellow;
    }
    </style>
</head>
<body>
    <table>
    <tr>
          <td>Выбери мене</td>
          <td>Чи мене</td>
          <td>Можна й мене</td>
          <td>Краще мене</td>
    </tr>
    <tr>
          <td>Выбери мене</td>
          <td>Чи мене</td>
          <td>Можна й мене</td>
          <td>Краще мене</td>
    </tr>
    <tr>
          <td>Нікого не слухай</td>
          <td>Краще сюди</td>
           <td>Я тут головний</td>
          <td>Готовий?</td>
    </tr>
    <tr>
```

```
            <td>Нікого не слухай</td>
            <td>Краще сюди</td>
              <td>Я тут головний</td>
            <td>Готовий?</td>
      </tr>
      </table>
</body>
</html>
```

JavaScript - Файл learning.js

```
let selected;
table = document.querySelector('table')
table.onmouseover = function(event) {
  let target = event.target;
  if (target.tagName === 'TD') {
      highlight(target);
      return;
  }
};
function highlight(td) {
  if (selected) {
    selected.classList.remove('colored');
  }
  selected = td;
  selected.classList.add('colored');
}
```

Тепер при наведенні миші на окрему комірку таблиці ми змінюємо її фоновий колір шляхом додавання нового класу, а коли комірка поза фокусом, то клас видаляється.

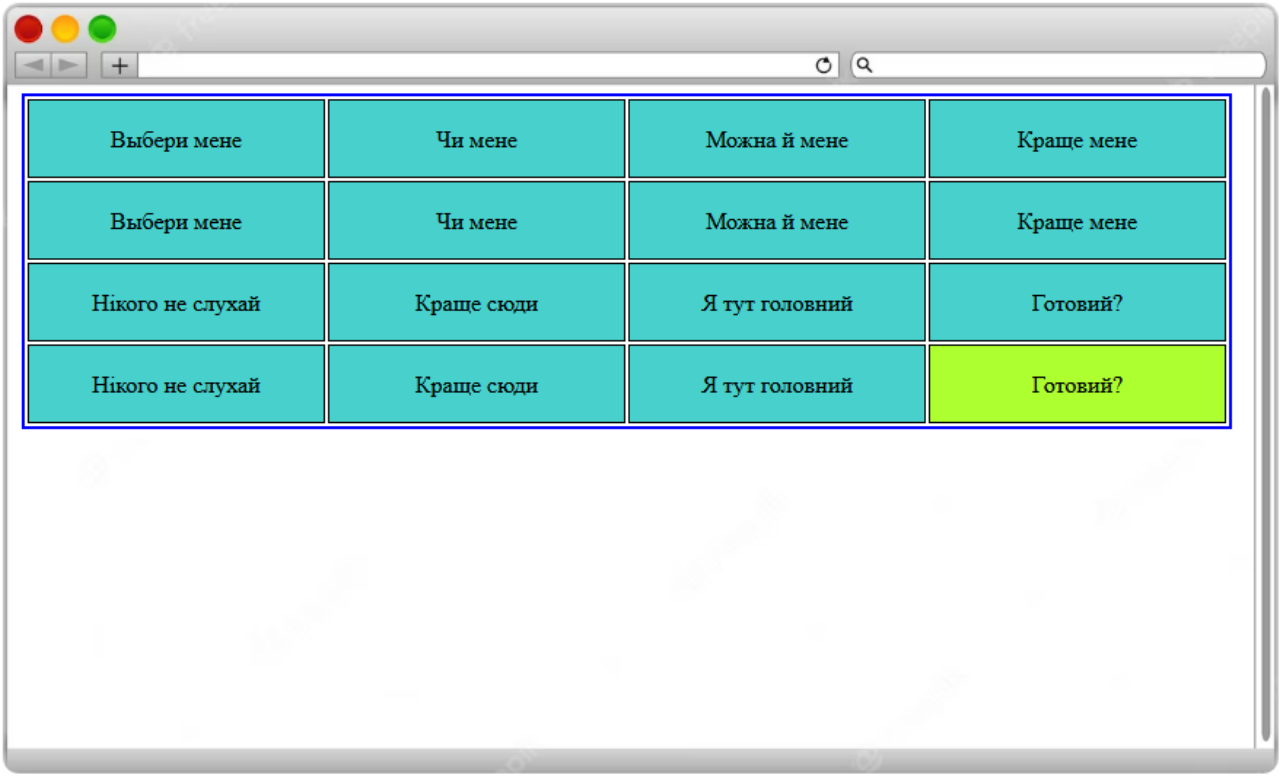


Рис. Зміна кольору комірки при наведенні миші

Без JavaScript не обходиться сучасний фронтенд. Ця мова програмування дозволяє керувати DOM-деревом за допомогою браузерного оточення. Можна вибирати необхідні елементи, модифікувати їх, змінювати кількість. На додаток, до кожного об'єкта не складно застосувати певні обробники подій, які суттєво розширять функціонал сайту та й користувачам стане приємніше взаємодіяти з ресурсом.

Запитання

1. У чому різниця між HTML-колекцією та колекцією вузлів?

Обидві колекції є живими, але у списку вузлів ми отримаємо не лише HTML-теги, а й текст із коментарями, тоді як у HTML-колекції представлені лише теги.

2. Для чого застосовується властивість classList ?

Властивість classList дозволяє отримати перелік всіх класів у певного елемента як окремо, так і в повному написанні (як воно записано в атрибуті class). Крім цього, клас можна змінити.

Приклад – HTML---

```
<body class="one two"></body>
```

Приклад – Консоль браузера---

```
> document.body.classList
< DOMTokenList(2) ["one", "two", value: "one two"]
> document.body.classList = ['uno']
> document.body.classList
< DOMTokenList ["uno", value: "uno"]
```

3. Перерахуйте можливі способи доступу до першого абзацу у наведеному прикладі. Всі вони знаходяться всередині тега <body>.

Приклад HTML---

```
<p id="paragraph" class="main-text">Первый абзац</p>
<p class="main-text" name="second">Второй абзац</p>
<p class="main-text">Третий абзац</p>
```

Кількість методів обмежена лише фантазією. Можна використовувати як навігаційні властивості, і вбудовані методи.

Приклад – Консоль браузера---

```
> document.getElementById('paragraph')
> document.body.firstElementChild
> document.body.children[0]
> document.getElementsByTagName('p')[0]
> document.getElementsByClassName('main-text')[0]
> document.querySelector('.main-text')
> document.querySelectorAll('.main-text')[0]
< <p id="paragraph" class="main-text">Перший абзац</p>
```

Завдання

Завдання 1

Є список посилань, як внутрішніх, і провідних сторонні ресурси. Зробіть для всіх сторонніх посилань верхнє підкреслення та зелений колір, а для сайту Google поставте лише колір червоного кольору. Посилання вигадайте самі з урахуванням умови.

Завдання 2

Усередині тегу розміром 700 x 500 px розташований інший елемент розміру 40 x 40 px. Досягніть наступного ефекту: коли користувач натискає ліву кнопку миші в будь-якій області зовнішнього тега, внутрішній елемент переміщається в це місце.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Обработчик событий</title>
  <script src="learning.js" defer></script>

  <style>
```

```
      #outer {
             width: 700px;
             height: 500px;
             border: 2px solid gold;
             background-color:khaki;
             position: relative;
             overflow: hidden;
             cursor: pointer;
      }
      #inner {
             position: absolute;
             left: 25px;
             top: 30px;
             width: 40px;
             height: 40px;
             transition: all 1.5s;
             background-color: mediumblue;
             border-radius: 50%;
      }
</style>
</head>
<body>
  <div id="outer">
    <div id="inner"></div>
  </div>
</body>

</html>
```

Рішення

Завдання 1

Підходів до рішення може бути декілька. Припустимо, що це посилання внутрішні сторінки представлені відносними шляхами.

Рішення – HTML

```
<a href="e.html">Внутрішня</a>
<a href="https://www.google.com/">Гугл</a>
<a href="https://www.site.com/">Зовнішня</a>
```

Рішення – JavaScript

```
let urls = document.querySelectorAll('a');
```

```
for (let url of urls) {
  let href = url.getAttribute('href');
  if (href.includes('http')) {
     url.style.color = 'green';
     url.style.textDecoration = 'overline';
  }
  if (href.includes('google.com')) {
     url.style.color = 'red';
     url.style.textDecoration = 'underline';
  }
}
```

Завдання 2

Так як внутрішній має абсолютне позиціонування, а зовнішній – відносне, координати меншого блоку розраховуватимуться щодо верхнього лівого кута батька. Необхідно також досягти того, щоб дочірній елемент не виїжджав за межі свого батька. Для плавності переміщення заданий перехід у розмірі 1,5 секунди.

Рішення – JavaScript

```
outer.onclick = function(event) {
     let outerCoords = this.getBoundingClientRect();
     let innerCoords = {
     top: event.clientY - outerCoords.top - outer.clientTop - inner.clientHeight / 2,
     left: event.clientX - outerCoords.left - outer.clientLeft - inner.clientWidth / 2
     };
     if (innerCoords.top < 0) innerCoords.top = 0;
     if (innerCoords.left < 0) innerCoords.left = 0;
     if (innerCoords.left + inner.clientWidth > outer.clientWidth) {
     innerCoords.left = outer.clientWidth - inner.clientWidth;
     }
     if (innerCoords.top + inner.clientHeight > outer.clientHeight) {
     innerCoords.top = outer.clientHeight - inner.clientHeight;
     }
     inner.style.left = innerCoords.left + 'px';
     inner.style.top = innerCoords.top + 'px';
}
```

Джерела інформації

1. JavaScript у веб-розробці https://smartiqa.ru/courses/web/lesson-7-js
2. Сучасний підручник з JavaScript https://uk.javascript.info/