ЛЕКЦІЯ 3

JavaScript: основи та взаємодія з DOM

Сайт реагує на дії користувача

Дисципліна «Веб-дизайн»
Викладач: Голенко Максим Юрійович

План лекції 3

01 **Підключимо JavaScript**

Зовнішній файл, defer і консоль браузера

02 **Прочитаємо простий код**

Змінні, умови, масиви, цикли, функції й об’єкти

03 **Змінимо DOM**

Пошук елемента, текст, CSS-клас і обробник події

04 **Розглянемо вебінтерфейс**

Картка, форма, збереження даних і запити до JSON

Як користуватися лекцією

01 **Послідовність**

Мова → DOM → події → форма → дані й запити → перевірка.

02 **Перед кодом**

Поняття → огляд методів → приклад і видимий результат.

03 **Таблиці за темами**

Рядки, масиви, DOM і події: потрібні засоби поруч із поясненнями.

04 **Навчальний результат**

Студент пояснює поведінку інтерфейсу та обирає потрібний метод.

БЛОК 1 · ПЕРШИЙ ЗАПУСК

JavaScript у браузері

Підключення скрипту, виконання коду та консоль

Лекція 3 · JavaScript
Навчальний проєкт: портфоліо

Де виконується JavaScript

Мова JavaScript

Код виконує рушій

У браузері код працює разом зі сторінкою. JavaScript також може виконуватися поза браузером.

API браузера

Доступ до сторінки

document, DOM та події дають коду змогу читати сторінку й реагувати на дії користувача.

На цій лекції працюємо у браузері з локальним портфоліо.

Підключаємо скрипт із defer

HTML

```
<head>
  <link rel="stylesheet"
        href="css/style.css">
  <script src="js/script.js"
          defer></script>
</head>
```

Стилі — у css/style.css, поведінка — у js/script.js.

src

Шлях до зовнішнього JS-файлу.

defer

Виконання після розбору HTML.

Порядок

DOM уже доступний нашому коду.

Консоль і перший запуск

JavaScript

```
console.log("Скрипт працює");

const title = "Портфоліо";
console.log(title);
```

console.log()

Виводить значення для перевірки.

Console

Вкладка інструментів розробника.

Помилка

Читаємо тип, файл і номер рядка.

Зміни у файлі зберігаємо, сторінку оновлюємо.

БЛОК 2 · ОСНОВИ МОВИ

Основи мови JavaScript

Значення → умови → масиви й цикли → функції та об’єкти

Лекція 3 · JavaScript
Навчальний проєкт: портфоліо

Змінні: const і let

const

Прив’язка не змінюється

Створює ім’я для значення. Після ініціалізації нове значення цьому імені не присвоюємо.

let

Можливе перепризначення

Створює змінну, значення якої може змінюватися під час виконання коду.

const використовуємо за замовчуванням; let — коли потрібна зміна.

Оголошення та зміна значення

JavaScript

```
const title = "Портфоліо";
let count = 1;

count = count + 1;
const active = false;

console.log(title, count, active);
```

Значення в JavaScript

```
Портфоліо
count = 2
active = false
```

Ім’я змінної посилається на значення. = виконує присвоєння.

Основні типи значень

01 **string — рядок**

Текст у лапках: "Портфоліо", "42"

02 **number — число**

Числа для обчислень: 42, 3.5

03 **boolean — логічне значення**

true або false: наприклад, чи виділено картку

04 **null та undefined**

null — явна відсутність; undefined — значення ще не визначене

Типи та перевірка через typeof

01 **Примітиви**

string, number, boolean, undefined, null, bigint і symbol.

02 **Об’єкти**

Об’єкти й масиви зберігають пов’язані дані; функції можна передавати як значення.

03 **typeof**

typeof "42" → "string"; typeof 42 → "number"; typeof (() => {}) → "function".

04 **Особливості**

typeof null → "object"; масив перевіряємо через Array.isArray(value).

Оператори та порівняння

01 **Присвоєння: =**

Записує значення у змінну: count = 2

02 **Обчислення: +, -, *, /**

Працюють із числами; + також поєднує рядки

03 **Порівняння: ===, !==, >, <**

Результат — true або false. === не перетворює типи

04 **Логіка: !, &&, ||**

Заперечення, «і», «або» для поєднання умов

Рядок і число — різні значення

JavaScript

```
const value = "42";
const number = Number(value);

console.log(value === 42); // false
console.log(number === 42); // true
console.log(number + 1); // 43
```

Число та рядок

```
value === 42: false
number === 42: true
number + 1: 43
```

Number() явно перетворює рядок на число.

Перетворення значень і NaN

01 **Явне перетворення**

Number("42") → 42; String(42) → "42"; Boolean(0) → false.

02 **Неочікуване додавання**

"2" + 1 → "21"; Number("2") + 1 → 3. Поле форми повертає рядок.

03 **Помилкове число**

Number("текст") → NaN; перевірка: Number.isNaN(value). NaN !== NaN.

04 **Порожній ввід**

Number("") → 0. Спочатку перевіряємо trim(), потім перетворюємо й перевіряємо число.

Числа, Math і Date

Засіб	Призначення / результат	Приклад
`Number.isFinite / isNaN`	Перевірки без перетворення	Number.isFinite(3) → true
`parseInt / parseFloat`	Читання числового префікса	parseInt("12px", 10) → 12
`toFixed(n)`	Форматування → рядок	(3.14159).toFixed(2) → "3.14"
`Math.round / floor / ceil`	Округлення / вниз / вгору	Math.floor(3.9) → 3
`Math.min / max / random`	Межі / випадкове [0, 1)	Math.max(3, 7) → 7
`Date.now / new Date()`	Мілісекунди / об’єкт дати	new Date().toLocaleDateString("uk-UA")

parseFloat("12abc") → 12; Number("12abc") → NaN. random не придатний для криптографії.

Рядки: текст, властивість і метод

01 **Створення**

Одинарні й подвійні лапки задають текст; зворотні лапки дозволяють ${вираз}.

02 **Незмінність**

Методи повертають новий рядок. Початковий текст сам не змінюється.

03 **Властивість**

text.length читаємо без (); результат — кількість одиниць UTF-16.

04 **Метод**

text.trim() викликаємо з (); аргументи уточнюють дію методу.

String: пошук у тексті

Засіб	Призначення / результат	Приклад
`includes(x)`	Наявність → boolean	"HTML CSS".includes("CSS") → true
`startsWith(x)`	Початок → boolean	"index.html".startsWith("index") → true
`endsWith(x)`	Закінчення → boolean	"index.html".endsWith(".html") → true
`indexOf(x)`	Перший індекс або -1	"banana".indexOf("a") → 1
`lastIndexOf(x)`	Останній індекс або -1	"banana".lastIndexOf("a") → 5
`search(regexp)`	Індекс збігу або -1	"a2".search(/\d/) → 1

Пошук чутливий до регістру. Позиції й індекси вимірюються в UTF-16.

String: доступ і фрагменти

Засіб	Призначення / результат	Приклад
`at(i)`	Одиниця UTF-16; можливий i < 0	"CSS".at(-1) → "S"
`charAt(i)`	Одиниця UTF-16 або ""	"CSS".charAt(0) → "C"
`slice(a, b)`	Фрагмент; b не входить	"JavaScript".slice(0, 4) → "Java"
`substring(a, b)`	Фрагмент; a і b може поміняти	"JavaScript".substring(4, 0) → "Java"
`split(sep)`	Розбиття → масив рядків	"HTML,CSS".split(",") → ["HTML", "CSS"]
`[Symbol.iterator]()`	Ітератор кодових точок	[..."A😀"].length → 2

slice підтримує від’ємні межі. substring перетворює від’ємні межі на 0.

String: очищення й регістр

Засіб	Призначення / результат	Приклад
`trim()`	Пробіли з обох країв	" CSS ".trim() → "CSS"
`trimStart() / trimEnd()`	Пробіли з одного краю	" CSS".trimStart() → "CSS"
`toLowerCase()`	Нижній регістр	"CSS".toLowerCase() → "css"
`toUpperCase()`	Верхній регістр	"css".toUpperCase() → "CSS"
`toLocaleLowerCase(loc)`	Регістр із правилами локалі	"I".toLocaleLowerCase("tr") → "ı"
`toLocaleUpperCase(loc)`	Регістр із правилами локалі	"i".toLocaleUpperCase("tr") → "İ"

Усі ці методи повертають новий рядок. Пробіли всередині trim не прибирає.

String: заміна й компонування

Засіб	Призначення / результат	Приклад
`replace(x, y)`	Заміна; рядок x — перший збіг	"a-a".replace("a", "b") → "b-a"
`replaceAll(x, y)`	Заміна всіх збігів	"a-a".replaceAll("a", "b") → "b-b"
`concat(...s)`	Поєднання рядків	"HTML".concat(" + CSS") → "HTML + CSS"
`repeat(n)`	Повторення n разів	"ha".repeat(2) → "haha"
`padStart(n, s)`	Доповнення зліва до довжини n	"7".padStart(3, "0") → "007"
`padEnd(n, s)`	Доповнення справа до довжини n	"7".padEnd(3, "0") → "700"

replaceAll із регулярним виразом потребує прапорця g; replace з /.../g теж замінює всі.

Регулярний вираз: шаблон пошуку

01 **Синтаксис**

/шаблон/прапорці; /\d+/g шукає послідовності цифр у тексті.

02 **Основні частини**

^ і $ — межі; [A-Z] — діапазон; \d — цифра; + — одне або більше повторень.

03 **Прапорці**

i — без урахування регістру; g — усі збіги; u — режим Unicode.

04 **Методи**

regexp.test(text) → boolean; text.match, replace і search використовують шаблон.

String: збіги, Unicode і порівняння

Засіб	Призначення / результат	Приклад
`match(regexp)`	Масив збігів або null	"a1b2".match(/\d/g) → ["1", "2"]
`matchAll(regexp)`	Ітератор збігів; regexp із g	[..."a1".matchAll(/\d/g)].length → 1
`localeCompare(s, loc)`	Порядок → число зі знаком	"a".localeCompare("b", "en") < 0
`normalize(form)`	Нормалізація Unicode	"é".normalize("NFC") → "é"
`isWellFormed()`	Немає непарних сурогатів?	"CSS".isWellFormed() → true
`toWellFormed()`	Непарні сурогати замінює на 🙂	"\uD800".toWellFormed() → "🙂"

localeCompare не гарантує саме -1 / 1. Кодова точка не завжди є видимим символом.

String: коди, значення, статичні методи

Засіб	Призначення / результат	Приклад
`charCodeAt(i)`	Числовий код одиниці UTF-16	"A".charCodeAt(0) → 65
`codePointAt(i)`	Кодова точка з позиції UTF-16	"😀".codePointAt(0) → 128512
`toString() / valueOf()`	Примітивний рядок з обгортки	new String("CSS").valueOf() → "CSS"
`String.fromCharCode(...n)`	Рядок з одиниць UTF-16	String.fromCharCode(65) → "A"
`String.fromCodePoint(...n)`	Рядок із кодових точок	String.fromCodePoint(128512) → "😀"
`String.raw`…`	Сирі частини шаблонного рядка	String.raw`a\nb` зберігає \n

Перші три — методи значення; останні три викликаються на String. length — властивість.

String: застарілі назви

Засіб	Призначення / результат	Приклад
`substr(start, length)`	Застарілий вибір фрагмента	Використовувати slice або substring
`trimLeft() / trimRight()`	Псевдоніми назв	Використовувати trimStart / trimEnd
`anchor() / link()`	Створення HTML із рядка	Створити <a> через DOM
`big() / small() / fontcolor()`	Застаріле HTML-оформлення	Оформлювати через CSS
`fontsize() / fixed() / bold()`	Застаріле HTML-оформлення	HTML-структура та CSS-клас
`blink / italics / strike / sub / sup`	Інші HTML-обгортки (методи)	Не використовувати для нового коду

Перелік містить сучасні власні методи String і застарілі назви; успадковані — у MDN.

Рядки та підготовка тексту

JavaScript

```
const message = "  Привіт!  ";
const clean = message.trim();

console.log(clean);
console.log(clean.length);
```

trim() повертає новий рядок. Пробіли всередині залишаються.

Лапки

Позначають рядкове значення.

trim()

Прибирає пробіли по краях.

length

Кількість одиниць UTF-16 у рядку.

Шаблонний рядок: текст із даними

JavaScript

```
const name = "Олена";
const count = 3;
const message =
  `Вітаємо, ${name}! Проєктів: ${count}`;

console.log(message);
```

Зворотні лапки

Починають і завершують шаблонний рядок.

${вираз}

Підставляє обчислене значення.

Результат

Вітаємо, Олена! Проєктів: 3

Шаблонні рядки полегшують формування підписів і повідомлень.

Умова if/else

if

Коли умова істинна

Виконує свій блок, якщо умова дає істинне значення. Наприклад, коли active дорівнює true.

else

Коли умова хибна

Виконує інший блок. else є необов’язковим.

Умова дає змогу вибрати дію відповідно до стану.

Умови: логічне перетворення

01 **Falsy**

false, 0, -0, 0n, "", null, undefined, NaN трактуються як хибні.

02 **Truthy**

Непорожні рядки, зокрема "false" і "0", а також [] і {} — істинні.

03 **Порівняння**

=== і !== не перетворюють типи; == може перетворювати їх.

04 **Гілки**

else if додає перевірку; ?: обирає одне з двох значень.

Довідка: оператори та вибір гілки

01 **Числа**

%, **, ++, --, +=, -=; % — остача, ** — піднесення до степеня.

02 **Коротке обчислення**

&& і || повертають операнд; || замінює будь-яке falsy, ?? — лише null / undefined.

03 **Доступ і вибір**

obj?.name → undefined за відсутності obj; condition ? a : b обирає значення.

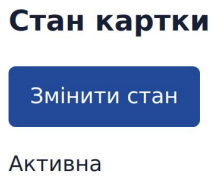
04 **Багато варіантів**

switch (value) зіставляє case; break завершує гілку; default — запасний варіант.

Стан визначає повідомлення

JavaScript

```
if (active) {
  console.log("Активна");
} else {
  console.log("Неактивна");
}
```



Натискання у живому прикладі змінює active. DOM розберемо далі.

Масиви: індекс, довжина, зміна

01 **Послідовність**

Масив об’єднує кілька значень: ["HTML", "CSS", "JavaScript"].

02 **Індекс**

Перший елемент має індекс 0; останній — length - 1.

03 **Читання й зміна**

items[0] читає значення; items[0] = "HTML5" змінює його.

04 **const і масив**

Можна змінювати елементи й додавати їх. Новий масив змінній не присвоюємо.

Array: додавання, вилучення й порядок

Засіб	Призначення / результат	Приклад
`push / pop`	Змінюють кінець масиву	push → нова довжина; pop → елемент
`unshift / shift`	Змінюють початок масиву	unshift → довжина; shift → елемент
`splice(a, n, ...x)`	Змінює; повертає вилучені	Вилучення або вставлення за індексом
`sort(fn) / reverse()`	Змінюють порядок у джерелі	nums.sort((a, b) => a - b)
`toSorted / toReversed`	Новий масив; джерело зберігають	nums.toSorted((a, b) => a - b)
`toSpliced / with(i, x)`	Новий масив зі зміною	[1, 2].with(0, 9) → [9, 2]

sort без функції порівнює як рядки. Нові методи перевіряємо на підтримку браузера.

Масив: кілька значень в одному місці

JavaScript

```
const technologies =
  ["HTML", "CSS", "JavaScript"];

console.log(technologies[0]); // HTML
console.log(technologies.length); // 3
```

Масив

Послідовність значень у [ ].

Індекс

Нумерація починається з нуля.

length

Кількість елементів масиву.

Масив зберігає значення; цикл дає змогу виконати дію для кожного з них.

Цикли: повторюємо дію

01 **Призначення**

Цикл виконує блок коду кілька разів без копіювання команд.

02 **for**

Початкове значення → перевірка умови → тіло → крок → знову перевірка.

03 **for...of**

По черзі отримує значення з масиву: наприклад, кожну технологію.

04 **Завершення**

for зупиняється, коли умова хибна; for...of — коли значення закінчилися.

Звичайний for: початок, умова і крок

JavaScript

```
const technologies =
  ["HTML", "CSS", "JavaScript"];

for (let i = 0; i < technologies.length; i++) {
  console.log(technologies[i]);
}
```

let i = 0

Початковий індекс задаємо один раз.

i < technologies.length

Перед кожним кроком перевіряємо умову.

i++

Збільшує і на 1 після тіла.

Результат у консолі: HTML, CSS, JavaScript — по одному значенню.

for...of: значення без ручного індексу

JavaScript

```
const technologies =
  ["HTML", "CSS", "JavaScript"];

for (const technology of technologies) {
  console.log(technology);
}
```

technologies

Масив, значення якого перебираємо.

technology

На кожній ітерації — наступне значення.

Вибір циклу

for — індекси; for...of — значення.

Той самий результат у консолі. Роботу з DOM додамо після його пояснення.

Довідка: while, break і continue

JavaScript

```
let count = 0;
while (count < 3) {
  count++;
  if (count === 2) continue;
  console.log(count); // 1, потім 3
}
// do { ... } while (condition);
```

while

Перевіряє умову перед кожним повторенням.

continue / break

continue переходить до наступного повторення; break завершує цикл.

do...while

Спочатку тіло, потім умова: хоча б одне виконання.

У циклі має змінюватися стан, від якого залежить завершення.

Функція: параметр і результат

01 **Функція**

Блок коду, який можна викликати за ім’ям

02 **Параметр**

Ім’я для вхідного значення всередині функції

03 **Виклик та аргумент**

getButtonLabel(true) запускає функцію й передає їй true

04 **return**

Повертає результат та завершує поточний виклик

Функція формує підпис кнопки

JavaScript

```
function getButtonLabel(active) {
  if (active) {
    return "Прибрати виділення";
  }
  return "Виділити картку";
}

getButtonLabel(true);
```

Результат getButtonLabel()

```
false: Виділити картку
true: Прибрати виділення
```

active — параметр; true у виклику — аргумент. Результат — рядок.

Область видимості: де доступне ім’я

JavaScript

```
const course = "Веб-дизайн";

function showCourse() {
  const message = "Вивчаємо " + course;
  console.log(message);
}

showCourse();
// console.log(message); // ReferenceError
```

Зовнішня змінна

course доступна всередині функції.

Локальна змінна

message доступна лише в showCourse().

Блок { }

let і const також обмежені своїм блоком.

Зовнішній код не бачить локальну змінну. Приклад помилки залишено коментарем.

Стрілкова функція та callback

JavaScript

```
const double = (number) => number * 2;
console.log(double(3)); // 6

const numbers = [1, 2, 3];
const doubled = numbers.map(double);
console.log(doubled); // [2, 4, 6]
```

У тілі { } результат потрібно повернути через return.

=>

Короткий запис функції; один вираз повертається автоматично.

callback

Функцію double передаємо іншому методу.

map()

Викликає функцію для елементів і створює новий масив.

Об’єкти: властивості та копіювання

01 **Пов’язані дані**

Об’єкт зберігає пари ключ: значення. Доступ: project.title або project["title"].

02 **Деструктуризація**

const { title, done = false } = project; const [first] = items;

03 **Spread**

const copy = { ...project }; const next = [...items, "JS"]; копія поверхнева.

04 **Методи Object**

Object.keys → ключі; values → значення; entries → пари; Object.hasOwn перевіряє власний ключ.

Об’єкт: пов’язані дані одного проєкту

JavaScript

```
const project = {
  title: "Портфоліо",
  technology: "JavaScript",
  completed: false
};

console.log(project.title);
project.completed = true;
```

Об’єкт { }

Об’єднує дані про один проєкт.

Властивість

Пара ім’я: значення; доступ через крапку.

const та об’єкт

Властивості змінюємо; прив’язку — ні.

Функцію в об’єкті називають методом. Далі працюватимемо з об’єктами DOM.

Метод об’єкта й значення this

JavaScript

```
const project = {
  title: "Портфоліо",
  describe() {
    return this.title;
  }
};
console.log(project.describe());
```

Стрілкова функція бере this із зовнішньої області, а не з об’єкта виклику.

Метод

Функція у властивості об’єкта.

this

У project.describe() посилається на project.

Спосіб виклику

Значення this залежить від того, як викликали звичайну функцію.

Методи масивів: обрати дію

01 **Пошук**

includes перевіряє наявність; find повертає перший потрібний елемент.

02 **Перетворення**

map створює нові значення; filter залишає елементи за умовою.

03 **Перебирання**

for...of читає значення; forEach виконує callback для кожного елемента.

04 **Зміна джерела**

push, pop, splice, sort змінюють масив; map і filter — ні.

Array: пошук і перевірка

Засіб	Призначення / результат	Приклад
`at(i) / includes(x)`	Значення / наявність	[2, 4].at(-1) → 4
`indexOf(x) / lastIndexOf(x)`	Індекс або -1	[2, 4, 2].lastIndexOf(2) → 2
`find(fn) / findIndex(fn)`	Елемент / індекс першого збігу	[2, 4].find(n => n > 2) → 4
`findLast / findLastIndex`	Пошук із кінця (методи)	[2, 4, 6].findLast(n => n < 6) → 4
`some(fn) / every(fn)`	Хоча б один / усі → boolean	[2, 4].every(n => n > 0) → true
`Array.isArray(value)`	Перевірка типу масиву	Array.isArray([]) → true

find повертає undefined за відсутності збігу; findIndex — -1.

Array: перетворення та зведення

Засіб	Призначення / результат	Приклад
`map(fn)`	Новий масив результатів	[1, 2].map(n => n * 2) → [2, 4]
`filter(fn)`	Новий масив відібраних	[1, 2].filter(n => n > 1) → [2]
`forEach(fn)`	Дії; результат undefined	items.forEach(item => console.log(item))
`reduce(fn, start)`	Зведення до одного значення	[1, 2].reduce((sum, n) => sum + n, 0) → 3
`join(s) / slice(a, b)`	Рядок / копія фрагмента	["HTML", "CSS"].join(', ')
`flat(n) / flatMap(fn)`	Сплощення / map + flat(1)	[[1], [2]].flat() → [1, 2]

forEach не замінює map. Для порожнього масиву reduce потребує початкового значення.

Масив проєктів: відбір і назви

JavaScript

```
const projects = [
  { title: "Сайт", done: true },
  { title: "Блог", done: false }
];
const titles = projects
  .filter(project => project.done)
  .map(project => project.title);
console.log(titles); // ["Сайт"]
```

filter()

Залишає проєкт, якщо callback повертає істинне значення.

map()

Кожний об’єкт перетворює на його назву.

Початкові дані

projects залишається масивом об’єктів.

Кожний етап ланцюжка повертає значення для наступного методу.

Доступ до елементів сторінки

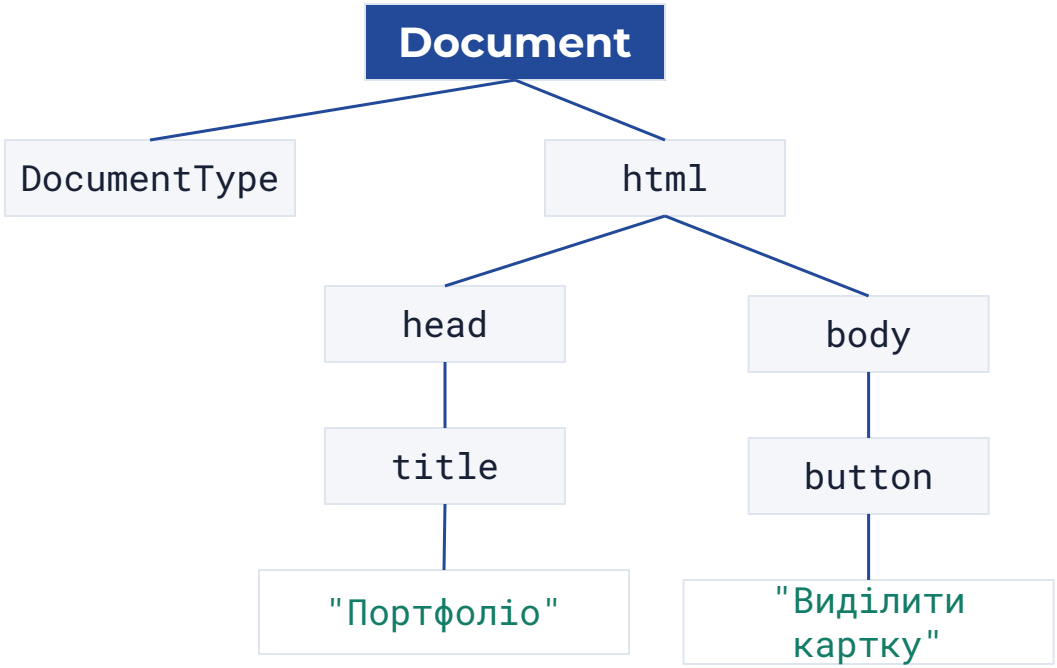
Браузер представляє документ як дерево об’єктів

Лекція 3 · JavaScript
Навчальний проєкт: портфоліо

Повертаємося до DOM-дерева

HTML

```html
<!DOCTYPE html>
<html lang="uk">
  <head>
    <title>Портфоліо</title>
  </head>
  <body>
    <button>Виділити картку</button>
  </body>
</html>
```



JavaScript знаходить вузол у DOM і змінює його властивості.
Структура та текст мають батьківські й дочірні зв’язки.

Об’єкт, властивість і метод

JavaScript

```
const status =
  document.querySelector("#project-status");

status.textContent = "Картку виділено";
```

Об’єкт

status посилається на DOM-елемент.

Властивість

textContent зберігає текст.

Метод

querySelector() виконує пошук.

Крапка дає доступ до властивості або методу об’єкта.

Як працює querySelector()

01 **Область пошуку**

document.querySelector() шукає в документі

02 **CSS-селектор**

"#toggle-project" — id; ".project-card" — клас

03 **Перший збіг**

Метод повертає перший відповідний елемент

04 **Немає збігу**

Результат null. Перевіряємо селектор і доступність DOM

DOM: знайти, змінити, створити

01 **Пошук**

querySelector — один елемент або null; querySelectorAll — колекція збігів.

02 **Зміна**

textContent задає текст; classList керує класами; hidden приховує елемент.

03 **Створення**

createElement створює елемент; append додає його; remove прибирає з DOM.

04 **Безпечний текст**

Для введеного користувачем тексту використовуємо textContent, а не innerHTML.

DOM: короткий довідник

Засіб	Призначення / результат	Приклад
`querySelector / querySelectorAll`	Перший елемент / NodeList	document.querySelector("#title")
`textContent / innerHTML`	Текст / розбір HTML	element.textContent = userText
`classList / hidden`	Класи / приховування	element.hidden = true
`getAttribute / setAttribute`	Читання / зміна атрибута	button.setAttribute("aria-pressed", "true")
`dataset / style`	data-* / вбудовані стилі	card.dataset.id; card.style.color = "blue"
`createElement / append / remove`	Створити / вставити / прибрати	list.append(document.createElement("li"))

Класи задають оформлення в CSS; JavaScript керує станом. innerHTML не очищує небезпечний ввід.

Знаходимо елементи портфоліо

JavaScript

```
const card = document.querySelector(
  "#interactive-project");
const button = document.querySelector(
  "#toggle-project");
const status = document.querySelector(
  "#project-status");
```

card

Картка, вигляд якої змінюємо.

button

Кнопка, яка запускає дію.

status

Окремий текстовий відгук.

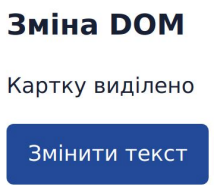
Ідентифікатори мають точно збігатися з атрибутами id у HTML.

Змінюємо текст через textContent

JavaScript

```
const status = document.querySelector(
  "#project-status");

status.textContent = "Картку виділено";
```



textContent записує текст у вибраний елемент.

Керуємо класами через classList

01 **add("is-selected")**

Додає клас до елемента

02 **remove("is-selected")**

Прибирає клас

03 **contains("is-selected")**

Перевіряє наявність класу та повертає boolean

04 **toggle("is-selected")**

Перемикає клас і повертає його новий стан: true або false

CSS описує стан, JavaScript його вмикає

CSS

```css
.project-card.is-selected {
  background: #eaf0ff;
  outline: 3px solid #224a98;
}
```

JavaScript

```javascript
card.classList.add("is-selected");

card.classList.remove("is-selected");

card.classList.toggle("is-selected");
```

Один CSS-клас визначає вигляд виділеної картки.

У метод передаємо ім’я класу без початкової крапки.

Створюємо список із масиву

JavaScript

```
const list = document.querySelector("#list");
for (const technology of technologies) {
  const item = document.createElement("li");
  item.textContent = technology;
  list.append(item);
}
```

Технології портфоліо

- HTML
- CSS
- JavaScript

У HTML є <ul id="list">. createElement() створює вузол, append() додає його.

Кілька карток: querySelectorAll()

JavaScript

```
const cards = document.querySelectorAll(
  ".project-card");

for (const card of cards) {
  card.classList.add("is-ready");
}
console.log(cards.length);
```

NodeList

Колекція елементів, не звичайний масив.

for...of

Виконує дію для кожної знайденої картки.

Немає збігів

Порожня колекція з length = 0.

querySelectorAll повертає статичну колекцію: нові вузли автоматично не додаються.

Реакція на дії користувача

Подія запускає обробник, який змінює DOM

Лекція 3 · JavaScript
Навчальний проєкт: портфоліо

Що таке подія

01 **Подія**

Сигнал про дію або зміну: click, input, submit

02 **Ціль**

Елемент, на якому виникла подія

03 **Обробник**

Функція, яку браузер викликає після події

04 **Підписка**

addEventListener() реєструє обробник для певного типу події

Об’єкт події: що сталося і де

01 **event.type**

Тип події: click, input, change, submit або keydown.

02 **target / currentTarget**

target — початкова ціль; currentTarget — елемент із поточним обробником.

03 **Спливання**

Багато подій переходять від дочірнього елемента до його предків.

04 **Стандартна дія**

preventDefault скасовує стандартну дію; stopPropagation зупиняє поширення.

Функція-обробник: передати чи викликати

JavaScript

```
function handleClick() {
  console.log("Натиснуто");
}

button.addEventListener(
  "click", handleClick
);
```

handleClick

Передаємо саму функцію без ().

handleClick()

Викликає функцію одразу в цьому місці.

Зворотний виклик

Браузер викликає її після click.

Підписка виконується зараз; тіло обробника — після події.

Додаємо обробник події

JavaScript

```
button.addEventListener(
  "click",
  function () {
    console.log("Натиснуто");
  }
);
```

button

Елемент, який слухаємо.

"click"

Назва події.

function () {...}

Функція-обробник для виклику.

Передаємо функцію браузеру. Вона виконається, коли виникне подія.

Подія, обробник і зміна DOM

01

Подія

Користувач активує кнопку

JavaScript

```
"click"
```

02

Обробник

Браузер викликає функцію

JavaScript

```
function () {
  // наша дія
}
```

03

DOM

Клас і текст отримують новий стан

JavaScript

```
card.classList
  .toggle("is-selected");
```

Готуємо кнопку та її CSS-стан

HTML · ФРАГМЕНТ

```
<article id="interactive-project"
         class="card project-card">
  <button id="toggle-project"
    type="button" aria-pressed="false">
    Виділити картку
  </button>
  <p id="project-status"></p>
</article>
```

CSS

```
.project-card.is-selected {
  background: #eaf0ff;
  outline: 3px solid #224a98;
}
```

Кнопка є окремим елементом усередині картки.

Повна версія HTML також містить aria-label та aria-live.

Кнопка змінює стан картки

JavaScript

```
button.addEventListener("click",
  function () {
    const active = card.classList
      .toggle("is-selected");
    button.textContent =
      getButtonLabel(active);
    status.textContent = getStatus(active);
  }
);
```

toggle()

Повертає нову наявність класу.

getButtonLabel()

Формує підпис кнопки.

getStatus()

Формує текстовий відгук.

Повний обробник у script.js також оновлює aria-pressed.

Делегування: один обробник для списку

JavaScript

```
list.addEventListener("click", event => {
  const button = event.target.closest(
    "button[data-id]");
  if (!button || !list.contains(button)) {
    return;
  }
  console.log(button.dataset.id);
});
```

Спливання

Натискання дочірньої кнопки доходить до list.

closest()

Знаходить відповідну кнопку або null.

Нові елементи

Один обробник працює і для доданих кнопок.

У HTML потрібен контейнер list із кнопками button[data-id].

Клавіатура та доступний стан

HTML + JavaScript

```
<button type="button"
  aria-label="Виділення картки"
  aria-pressed="false">
  Виділити картку
</button>

// в обробнику click:
button.setAttribute(
  "aria-pressed", String(active));
```

button

Працює через Enter і пробіл.

aria-pressed

Повідомляє стан перемикача.

aria-label

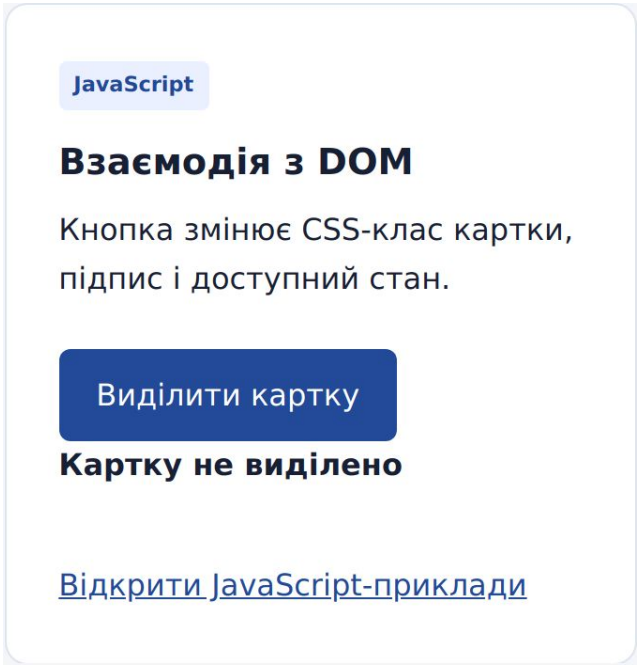
Зберігає сталу доступну назву.

Видимий підпис змінюється. Доступна назва й стан залишаються узгодженими.

Картка до та після натискання

Клас, підпис і доступний стан змінюються разом.

ПОЧАТКОВИЙ СТАН



КАРТКУ ВИДІЛЕНО



Enter, пробіл або натискання повертають картку в попередній стан.

Значення полів та локальний відгук

Вбудована перевірка HTML і власний обробник JavaScript

Лекція 3 · JavaScript
Навчальний проєкт: портфоліо

Як працює наша форма

01 **HTML-поля**

label, type="email", required і minlength задають основу

02 **value**

Властивість поля містить його поточне значення

03 **submit**

Подія форми після успішної вбудованої перевірки

04 **Локальний відгук**

Обробник показує повідомлення на сторінці

Події та поля форми: довідник

Засіб	Призначення / результат	Приклад
`click / keydown`	Активація / клавіша	event.key === "Escape"
`input / change`	Редагування / підтверджена зміна	input.addEventListener("input", handler)
`submit / preventDefault`	Надсилання / стандартна дія	form.addEventListener("submit", handler)
`value / checked / selectedOptions`	Рядок / стан / вибрані option	checkbox.checked → boolean
`checkValidity / reportValidity`	Перевірити / показати помилки	form.reportValidity()
`FormData / removeEventListener`	Дані форми / зняти обробник	new FormData(form); потрібен name

Зняти слухача можна за тією самою функцією та capture. Невибрані checkbox / radio не входять у FormData.

Читаємо значення поля

JavaScript

```
const emailInput =
  document.querySelector("#email");
const messageInput =
  document.querySelector("#message");

const email = emailInput.value;
const message = messageInput.value.trim();
```

value

Поточний текст input або textarea.

trim()

Прибирає крайні пробіли.

Момент читання

Усередині обробника беремо нове значення.

Значення текстового поля є рядком, навіть якщо користувач увів цифри.

checkbox, radio і select у JavaScript

JavaScript

```
const consent = document.querySelector(
  "#consent").checked;
const format = document.querySelector(
  'input[name="format"]:checked');
const topic = document.querySelector(
  "#topic").value;
console.log(consent, format?.value, topic);
```

checkbox.checked

Логічний стан true / false.

radio:checked

Обраний елемент; якщо не обрано — null.

select.value

Значення вибраного option.

?. читає властивість лише якщо об’єкт не null / undefined.

submit і preventDefault()

JavaScript

```
const form =
  document.querySelector("#contact-form");

form.addEventListener("submit",
  function (event) {
    event.preventDefault();
  }
);
```

submit

Слухаємо форму, а не лише кнопку.

event

Об’єкт поточної події.

preventDefault()

Скасовує стандартне надсилання.

Метод самостійно не надсилає й не зберігає дані.

Додаткова перевірка тексту

JavaScript

```
function getFeedback(message) {
  if (message.length < 10) {
    return "Додайте щонайменше 10 символів.";
  }
  return "Дані перевірено локально. " +
    "Надсилання ще не підключено.";
}
```

message

Уже очищений через trim() рядок.

if / return

Короткий текст отримує пояснення.

Результат

Функція повертає повідомлення.

Для перевірки викликаємо getFeedback(messageInput.value.trim()).

Показуємо відгук після submit

JavaScript

```
form.addEventListener("submit",
  function (event) {
    event.preventDefault();
    const message =
      messageInput.value.trim();
    feedback.textContent =
      getFeedback(message);
  }
);
```

messageInput

Читаємо значення під час дії.

getFeedback()

Перевіряє текст і формує відгук.

feedback

Окремий елемент aria-live="polite".

У повній версії подія input прибирає застарілий відгук.

Що означає результат демонстрації

Уже працює

Локальна взаємодія

Браузер перевіряє поля, JavaScript перевіряє очищений текст і показує відгук.

Потрібне продовження

Серверне надсилання

Для доставки повідомлень потрібен обробник або сервіс. Він також перевіряє отримані дані.

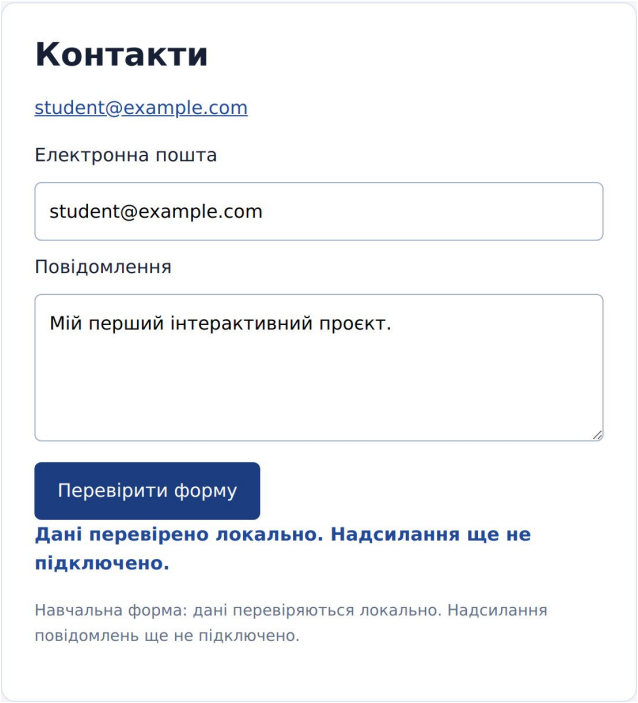
Повідомлення про перевірку не означає доставку листа.

Готова форма у портфоліо

HTML + JavaScript

```
<p id="form-feedback"
   class="form-feedback"
   aria-live="polite">
</p>

// після локальної перевірки:
feedback.textContent =
  getFeedback(message);
```



Відгук з’являється на сторінці без переходу на іншу адресу.

Дані й можливості браузера

01 **JSON**

Текстовий формат обміну даними: parse читає, stringify перетворює.

02 **localStorage**

Зберігає рядки між відкриттями сторінки; об’єкти записуємо через JSON.

03 **Асинхронні дії**

Таймери й fetch завершуються пізніше. Promise та await описують очікування.

04 **Модулі**

export та import розділяють код на файли. У HTML потрібен type="module".

Обробка помилок: try і catch

01 **try**

Виконує код, який може викинути виняток, наприклад JSON.parse.

02 **catch (error)**

Отримує виняток і дозволяє показати зрозуміле повідомлення.

03 **throw / finally**

throw створює виняток; finally виконує завершальні дії.

04 **Межі**

catch не виправляє логіку автоматично; синтаксична помилка не дає розібрати цей самий скрипт.

JSON: перетворення й перевірка

JavaScript

```
const text = '{"title":"Портфоліо"}';
try {
  const project = JSON.parse(text);
  console.log(project.title);
  console.log(JSON.stringify(project));
} catch (error) {
  console.error(error.message);
}
```

JSON.parse()

Текст JSON перетворює на значення.

JSON.stringify()

Серіалізує підтримувані дані в текст JSON.

catch

Перехоплює помилку некоректного JSON.

JSON не зберігає функції; ключі й рядки записуються в подвійних лапках.

localStorage: збереження налаштування

JavaScript

```
try {
  localStorage.setItem("theme", "dark");
  const theme =
    localStorage.getItem("theme") ?? "light";
  console.log(theme);
} catch (error) {
  console.warn("Сховище недоступне");
}
```

setItem / getItem

Запис рядка / читання рядка або null.

removeItem

Прибирає один запис за його ключем.

Область

Налаштування зберігається для origin.

Перевіряти через HTTP-сервер; для file:// поведінка не гарантується.

Асинхронність: результат приходить пізніше

01 **Послідовний код**

Поточні синхронні інструкції виконуються до завершення свого виклику.

02 **Promise**

Описує майбутній результат: pending → fulfilled або rejected.

03 **async / await**

async повертає Promise; await призупиняє цю async-функцію до результату.

04 **Порядок**

Завершені Promise обробляються через мікрозадачі; таймери — через чергу задач.

Таймер: виконати дію пізніше

JavaScript

```
console.log("A");
const timer = setTimeout(() => {
  console.log("B");
}, 0);
console.log("C");
// порядок: A, C, B
// clearTimeout(timer) скасовує таймер
```

setTimeout()

Планує один виклик; затримка не є точним часом запуску.

setInterval()

Планує повторення; clearInterval зупиняє їх.

Крок анімації

requestAnimationFrame(): виклик перед кадром.

Затримка 0 не означає негайного виконання в поточному виклику.

fetch: отримати JSON із сервера

JavaScript

```
async function loadProjects() {
  const response = await fetch(
    "./projects.json");
  if (!response.ok) {
    throw new Error("HTTP " + response.status);
  }
  return await response.json();
}
```

fetch()

Повертає Promise з Response.

response.ok

Перевіряє успішний HTTP-статус.

response.json()

Асинхронно читає тіло JSON.

Запускати через HTTP-сервер; projects.json має існувати. Зовнішній сервер потребує CORS.

Асинхронний виклик і повідомлення

JavaScript

```
async function showProjects() {
  try {
    const projects = await loadProjects();
    console.log(projects);
  } catch (error) {
    console.error(error.message);
  }
}
showProjects();
```

Передумова

loadProjects оголошена на попередньому слайді.

try / catch

Обробляє мережеві, HTTP та JSON-помилки.

Інтерфейс

Додатково показує стан завантаження, результат або помилку.

fetch не відхиляє Promise лише через 404 / 500: статус перевіряємо самостійно.

Модулі: розділити код між файлами

JavaScript

```
// utils.js
export function double(number) {
  return number * 2;
}

// main.js
import { double } from "./utils.js";
console.log(double(3));
```

export

Робить функцію доступною іншому модулю.

import

Імпортує її за точним шляхом.

HTML

<script type="module" src="main.js"></script>

Модулі мають власну область видимості й строгий режим. Перевіряємо через HTTP-сервер.

БЛОК 6 · НАЛАГОДЖЕННЯ

Пошук помилок і завершення проєкту

Підключення файлу, консоль, селектор і поточний стан

Лекція 3 · JavaScript
Навчальний проєкт: портфоліо

Типові помилки JavaScript

01 **SyntaxError**

Незакрита лапка або дужка порушує синтаксис

02 **ReferenceError**

Ім’я змінної не оголошено або недоступне в цій області

03 **TypeError після null**

querySelector() нічого не знайшов, а код звернувся до властивості

04 **Помилка логіки**

Код виконується, але змінює не той елемент або не той стан

Налагоджуємо приклад у браузері

01 **Підключення**

Перевіряємо шлях js/script.js та завантаження файлу

02 **Console**

Читаємо першу помилку, назву файлу та номер рядка

03 **DOM та селектор**

У Elements перевіряємо id; у консолі — результат querySelector()

04 **Виконання**

Ставимо точку зупину в обробнику, натискаємо кнопку й читаємо active

Межі довідника й подальші теми

01 **Уже можна застосувати**

Значення, умови, цикли, функції, колекції, DOM, події, форми, сховище й запити.

02 **Поглиблення мови**

Замикання, прототипи, класи, Map / Set, ітератори, генератори й детальна модель виконання.

03 **Подальша веброзробка**

Серверна перевірка, API, безпека, автоматизовані тести й інструменти збирання.

04 **Документація**

MDN: JavaScript Reference, String, Array, DOM і Fetch; перевіряти сумісність браузерів.

Від структури до інтерактивності

Завершений проєкт

HTML + CSS + JavaScript

Семантична структура, адаптивне оформлення, події та зміна стану DOM.

Результат огляду

Дані та поведінка

Рядки, масиви, об’єкти, події, форми, JSON, сховище та асинхронні запити.

Ми можемо пояснити кожен крок: подія, обробник, зміна DOM.