Лабораторна робота №7

JavaScript: змінні, події, введення та виведення даних

Вступ

Ви вже вмієте програмувати: змінні, умови, цикли, функції та класи знайомі вам з C# чи C++. Тому в цій роботі ми не вивчаємо програмування з нуля, а дивимося, **чим JavaScript відрізняється** від звичних мов і як він «оживляє» веб-сторінку.

Будь-яка гра працює за одним циклом: **ввід → зміна стану → відображення**. Гравець натискає клавішу (подія), змінюється змінна (позиція, HP, золото), і екран перемальовується. У браузері все так само:

- **HTML-елементи** — це «сцена»: кнопки, поля, блоки, які бачить гравець;
- **змінні JavaScript** — це «стан гри»: числа й рядки, з якими працює логіка;
- **події** (`click`, `input`, `keydown`…) — це «система вводу», яка з'єднує сцену зі станом.

Мета лабораторної — навчитися зв'язувати ці три шари: **прийняти** дані від користувача, **обробити** їх у змінних та **вивести** результат на сторінку.

Теоретичні відомості

https://uk.javascript.info/alert-prompt-confirm

JavaScript очима C#/C++ розробника

Більшість помилок у перших скриптах виникає не через логіку, а через те, що JS поводиться інакше, ніж звичні мови. Ось головні відмінності:

Звично (C# / C++)	Як у JavaScript	Чому це важливо
int, string, float	Динамічна типізація: тип має значення, а не змінна	`prompt()` та `input.value` **завжди** повертають рядок
"5" + 3 → помилка компіляції	`"5" + 3` → `"53"`, але `"5" * 3` → `15`	Перед обчисленнями перетворюйте: `Number(str)`
7 / 2 → 3 (ціле ділення)	`7 / 2` → `3.5`	Ціле ділення: `Math.floor(a / b)`, остача: `a % b`
==	`===` (без приведення типів)	`0 == ""` → `true`, тому завжди `===`
null	`null`, `undefined` та `NaN`	`Number("abc")` → `NaN`; перевірка: `Number.isNaN()`
var / auto	`let` (змінна), `const` (константа)	`var` у сучасному коді не використовуємо
delegate, UnityEvent, event Action	Функція як значення: `addEventListener("click", fn)`	Обробник події — звичайна функція-колбек
$"HP: {hp}"	Шаблонні рядки: `` `HP: ${hp}` ``	Зручно збирати текст для виведення
Update() кожен кадр	Подієва модель: код чекає на дії користувача	Ігровий цикл (requestAnimationFrame) — у наступних роботах

Базовий каркас

Усі завдання будуються за однією схемою: знайти елемент → підписатися на подію → у обробнику прочитати значення, обчислити й вивести результат.

```
//.html
<input type="text" id="heroName">
<button id="createBtn">Створити</button>
<p id="output"></p>

//скрипт в кінці html
<script src="script.js" defer></script>
```

```
// script.js
// 1. знайти елементи
const nameInput = document.getElementById('heroName');
const output    = document.getElementById('output');

// 2. підписатися на подію
nameInput.onclick = showInfo();
function showInfo(){
    const name = nameInput.value.trim();          // 3. прийняти
    output.textContent = `Вітаємо, ${name}!`;   // 4. вивести

}
```

Завдання 1. Базова математика

Реалізувати можливість онлайн покупки. Користувач вводить суму грошей у гаманці та ціну одного бургера.

Програма виводить, скільки бургерів може купити користувач і скільки здачі в нього залишиться.

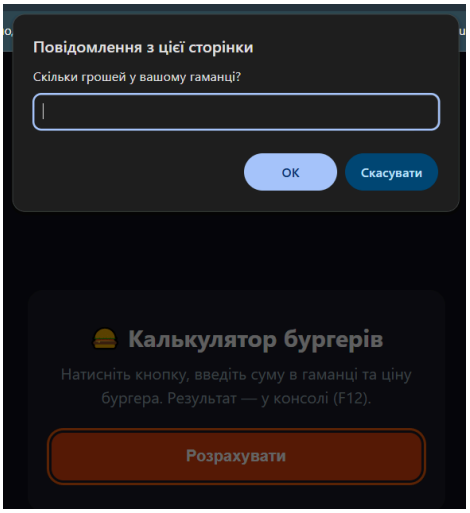
Нові JS-можливості

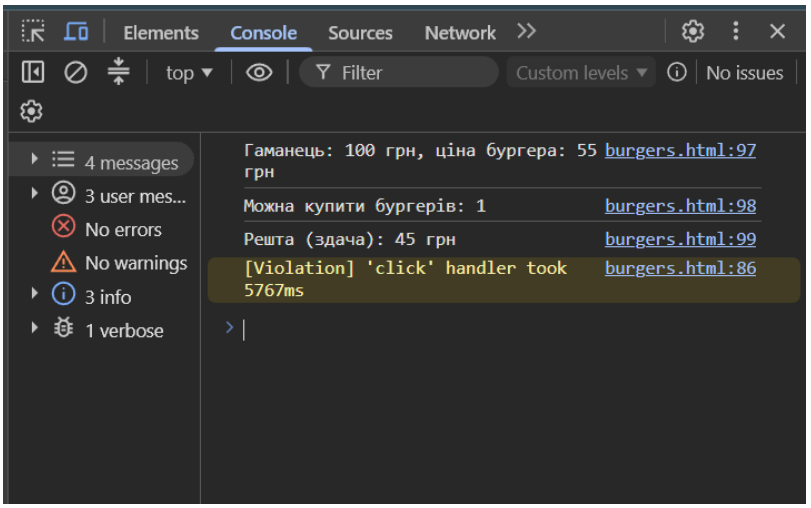
Конструкція	Що робить
`prompt(text)`	Показує діалог з полем вводу. Повертає **рядок**, або `null`, якщо натиснуто «Скасувати».
`alert(text)`	Показує повідомлення і зупиняє виконання скрипту, доки користувач не натисне ОК.
`Number(str)`	Перетворює рядок у число. `"abc"` → `NaN`, але `""` → `0` (!).
`Number.isInteger(x)`	Перевіряє, чи є значення цілим числом.
`Math.floor(x), %`	Округлення вниз та остача від ділення — замінюють цілочисельне ділення.
`console.log()`	Виводить значення в консоль DevTools (F12) — головний інструмент налагодження.

Вимоги

- Натискання «Скасувати» в будь-якому діалозі коректно завершує програму без помилок
- Зрозуміле повідомлення про помилку: некоректний ввід (літери, порожній рядок, від'ємні числа, ціна 0)
- Результат виводиться через `alert` та дублюється в `console.log`

Приклад результату:





Завдання 2. Створення героя

Користувач вводить ім'я героя в текстове поле. При натисканні на кнопку на сторінці з'являється привітання. Під полем у реальному часі показується лічильник символів.

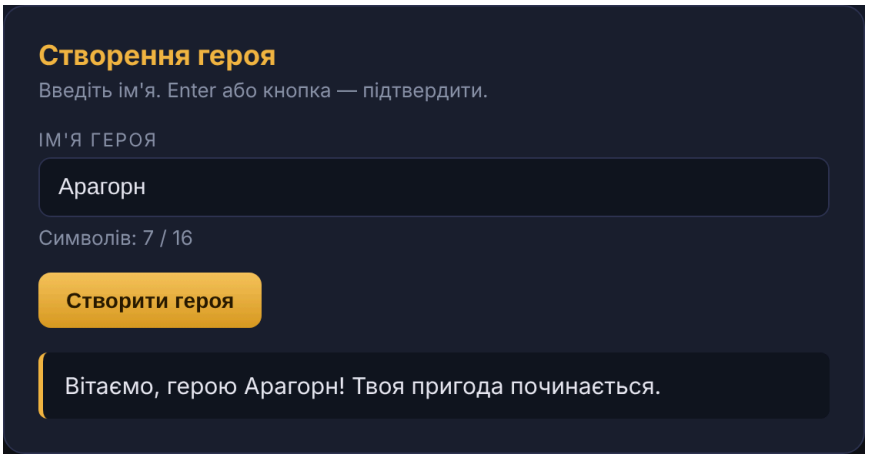
Нові JS-можливості

Конструкція	Що робить
`document.getElementById(id)`	Знаходить елемент за **`id`** і повертає посилання на нього (або **`null`**).
`el.onclick = fn`	Підписує функцію **`fn`** на подію **`el`**. Аналог **`button.onClick.AddListener()`** в Unity.
`input.value`	Поточний текст у полі вводу (рядок).
`el.textContent`	Записує текст в елемент. Безпечно: HTML-теги не виконуються.
`подія input`	Спрацьовує **на кожен символ**, на відміну від **`change`** (після втрати фокусу).

Вимоги

- Ім'я від 2 до 16 символів, пробіли по краях обрізаються (**`trim()`**)
- Лічильник символів оновлюється при кожному натисканні

Приклад результату



Завдання 3. Конструктор арени

Задано розміри прямокутної арени в тайлах. При натисканні на кнопку обчисліть довжину огорожі (периметр) та площу арени, а також її розмір у пікселях, якщо один тайл має 32×32 px.

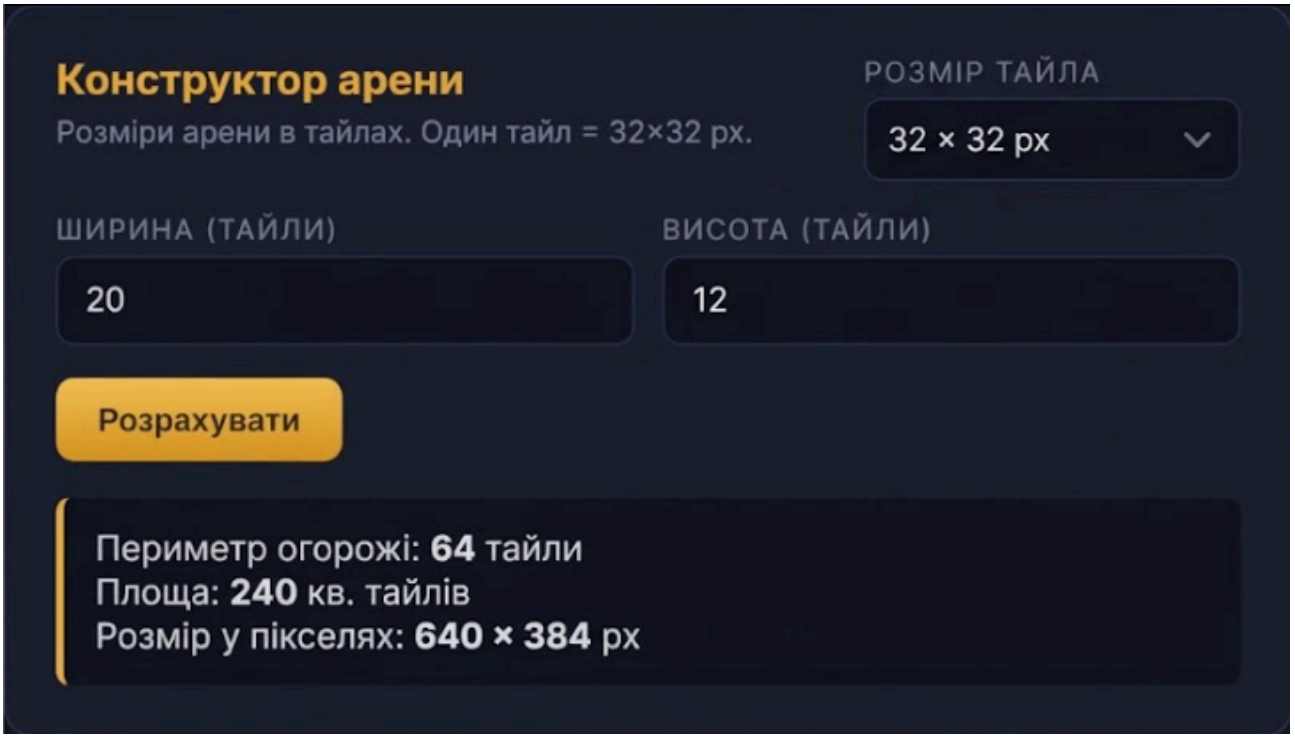
Нові JS-можливості

Конструкція	Що робить
`el.innerHTML`	Записує в елемент **HTML-розмітку** (теги `<b>`, ` ` тощо будуть відображені).
`textContent vs innerHTML`	`textContent` — для тексту від користувача, `innerHTML` — лише для вашої власної розмітки.
`<input type="number">`	Поле для чисел, але **`input.value` все одно рядок**! Альтернатива: `input.valueAsNumber`.
`const TILE = 32`	Константа: незмінне число записане в одному місці.
`Number.isFinite(x)`	Перевіряє, що значення — звичайне число (не `NaN` і не `Infinity`).

Вимоги

- Результат: кілька рядків з оформленими значеннями
- Валідація: порожні поля, нуль, від'ємні числа, помилки при введенні
- Розмір тайла заданий через `<select>` (16 / 32 / 64)

Приклад результату



Завдання 4. Скін героя

На сторінці є блок-«персонаж» та поле вибору кольору. При виборі кольору блок заливається ним, а всередині з'являється шістнадцяткове значення кольору. Повзунок змінює масштаб персонажа.

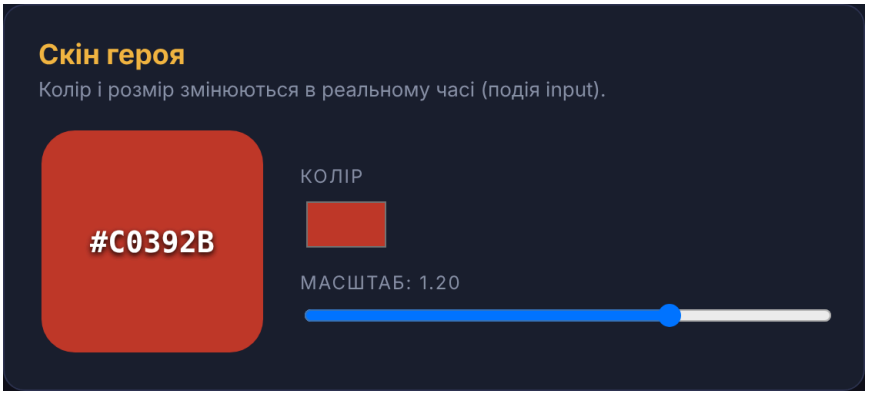
Нові JS-можливості

Конструкція	Що робить
`<input type="color">`	Поле вибору кольору; **`value`** — рядок вигляду **`"#5aa3ef"`**.
`<input type="range">`	Повзунок з атрибутами **`min`**, **`max`**, **`step`**.
`el.style.backgroundColor`	Inline-стиль елемента. CSS-властивості пишуться в **camelCase**: **`background-color`** → **`backgroundColor`**.
`el.style.transform`	Наприклад, `` `scale(${k})` `` — зміна масштабу без зміни розмітки.

Вимоги

- Колір змінюється в реальному часі (подія **`input`**), hex виводиться у верхньому регістрі
- Масштаб від 0.5 до 1.5, поточне значення показане поруч з повзунком
- Накладіть PNG зображення(з прозорим фоном) для зміни забарвлення елементів
- Додайте декілька полів зміни кольору(зміна фону, кольору окремих елементів та ін)

Приклад результату



Завдання 5*. Тест реакції

Блок має три стани: очікування, старт відліку, очікування події. Клік запускає тест, а блок стає червоним. Через випадковий час (1–3 с) він стає зеленим, і гравець має клікнути якнайшвидше. Виводиться час реакції в мілісекундах та найкращий результат. Клік на червоному — фальстарт.

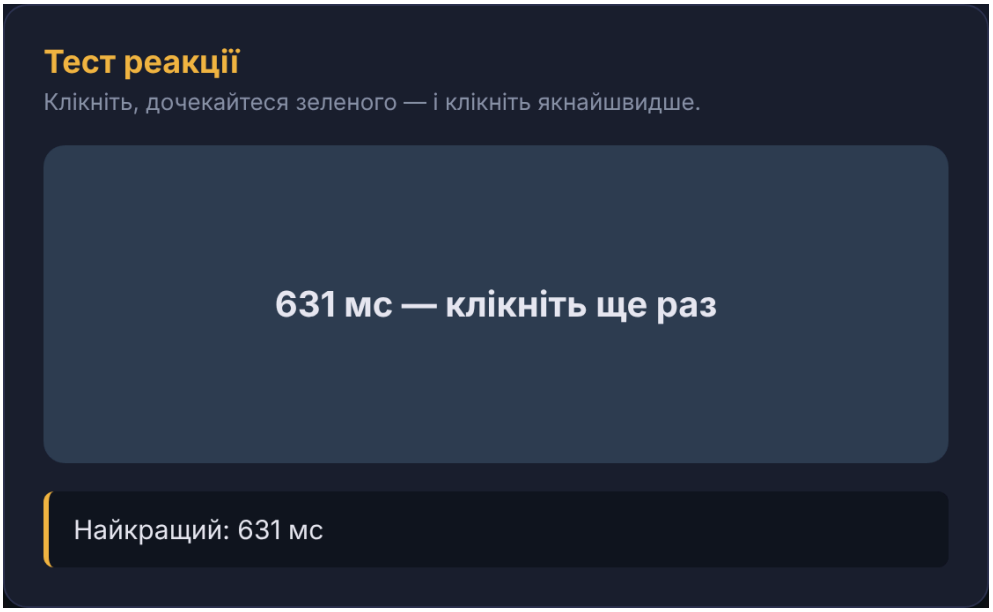
Нові JS-можливості

Конструкція	Що робить
`setTimeout / clearTimeout`	Одноразовий відкладений виклик та його скасування.
`performance.now()`	Точний час у мілісекундах — для вимірювання інтервалів.
`let phase = "idle"`	Скінченний автомат: одна змінна визначає поведінку (як стани в Unity Animator).
`Math.random()`	Випадкове число від 0 до 1

Вимоги

- Три стани (очікування, «чекай», «клікай»)
- Фальстарт скасовує таймер і повідомляє гравця
- Найкращий результат
- Середнє за 5 спроб

Приклад результату

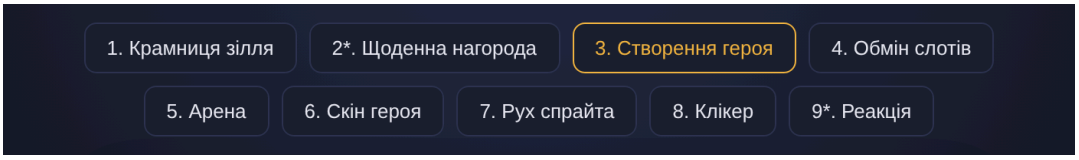


Завдання 6. Головне меню та звіт

Вимоги

- Головне меню: кнопка для кожного завдання; кожне завдання — окрема функція
- Валідація введених значень у всіх полях
- Структура проєкту: **`index.html`**, **`style.css`**, **`script.js`**

Приклад результату



Звіт має містити

1. Для кожного завдання: скріншот результату та ключовий фрагмент коду з поясненням
2. Промпти, які ви використовували
3. Висновки

Контрольні питання

1. Чому `"5" + 3` дає `"53"`, а `"5" - 3` дає `2`?
2. Чим відрізняються `==` та `===`? Наведіть приклад, де результати різні.
3. Що повертає `prompt()`, якщо натиснути «Скасувати»? А якщо нічого не ввести й натиснути ОК?
4. Чим відрізняються `textContent` та `innerHTML`? Коли `innerHTML` небезпечний?
5. Чим відрізняються події `click`, `input`, `change` та `keydown`?
6. Чому для ігрового керування краще використовувати `e.code`, а не `e.key`?
7. Чому стан гри краще зберігати в змінних, а не читати зі сторінки?
8. Що станеться, якщо підключити скрипт у `<head>` без атрибута `defer`?
9. Чому `addEventListener` кращий за атрибут `onclick` у HTML?