МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКИЙ КОЛЕГІУМ»
імені Т. Г. ШЕВЧЕНКА

Г. Ю. Цибко
Ю. В. Горошко
А. О. Костюченко

Основи програмування мовою JavaScript

Чернігів 2026

УДК 004.438 (075.8)

ORCID https://orcid.org/0000-0002-1861-3003
ORCID https://orcid.org/ 0000-0001-9290-7563
ORCID https://orcid.org/0000-0002-6178-6444

Ц56 Цибко Г.Ю., Горошко Ю.В., Костюченко А.О.
Основи програмування мовою JavaScript: навчальний посібник. Чернігів: НУЧК, 2026, 205 с.

Подано теоретичний матеріал, практичні завдання, інструкції та рекомендації до проведення лекцій, практичних занять і лабораторних робіт з основ програмування мовою JavaScript. Тематика посібника включає основи вебтехнологій, основи алгоритмізації в контексті використання JavaScript, огляд структур даних, основи роботи з об’єктною моделлю вебдокумента, створення інтерактивних вебдокументів. Посібник складено з урахуванням досвіду викладання програмування студентам освітніх програм „Комп’ютерні науки”та „Середня освіта (інформатика)”. Може бути використаний науково-педагогічними працівниками ЗВО, ЗФПО та ЗЗСО, здобувачами середньої та вищої освіти, а також у процесі самостійного вивчення основ програмування.

Рецензенти:
Пецко Василь Іванович - кандидат технічних наук, доцент
кафедри інформаційних управляючих систем та технологій ДВНЗ «Ужгородський Національний університет»

Горчинський Сергій Володимирович - кандидат педагогічних наук, доцент
кафедри технологічної освіти та інформатики
Національного університету «Чернігівський
колегіум» імені Т.Г.Шевченка

Мехед Дмитро Борисович, кандидат педагогічних наук, доцент
кафедри кібербезпеки та математичного моделювання
Національного університету «Чернігівська політехніка»

Рекомендовано до друку вченою радою Національного університету «Чернігівський колегіум»
імені Т.Г.Шевченка, протокол № __ від __ ___ 2026 р.

Зміст

ВСТУП

Навчання програмування є найважливішим компонентом формування інформаційно-цифрової компетентності та фахової підготовки у галузі інформаційних технологій в закладах освіти різних рівнів. Принцип академічної свободи дає викладачеві можливість вибору мови програмування для початківців, для поглибленого вивчення або для досягнення специфічних фахових цілей. В усіх зазначених аспектах популярною упродовж багатьох років залишається мова JavaScript.

У навчанні інформатики мова програмування JavaScript відіграє двоїсту роль. Вона може бути ефективно використана як основа для опанування основ алгоритмізації, типів і структур даних, різних парадигм програмування, зокрема об'єктноорієнтованого підходу. Крім того, JavaScript є фундаментом вебтехнологій, що в сучасному світі стали основою цифровізації усіх сфер суспільного життя. Знання принципів функціонування і навички роботи з вебтехнологіями набувають особливої значущості для формування інформаційно-цифрової компетентності здобувачів освіти.

Простота синтаксису JavaScript, наявність в загальному доступі великої кількості матеріалів для самоосвіти, широкий вибір редакторів коду, наявність спільноти для підтримки розробників та орієнтованість на вебтехнології сприяють мотивуванню здобувачів освіти до навчання програмування та подальшого розвитку в галузі інформаційних технологій.

Структура і зміст навчального посібника складені на основі досвіду викладання курсу «Програмування. Мова JavaScript» студентам природничо-математичного факультету Національного університету «Чернігівський колегіум» імені Т.Г.Шевченка (освітні програми підготовки бакалаврів «Середня освіта (Інформатика)» та «Комп'ютерні науки»). Вивченню наведеного матеріалу має передувати ознайомлення з основами HTML і CSS. Викладення основ алгоритмізації в посібнику дозволяє розглядати JavaScript як першу мову програмування.

Тематика посібника включає основи вебтехнологій, основи алгоритмізації в контексті використання JavaScript, огляд структур даних, основи роботи з об'єктною моделлю вебдокумента, створення інтерактивних вебдокументів. До посібника не включені питання, пов'язані з визначеними редакторами та фреймворками JavaScript, як такі, що значною мірою залежать від умов організації освітнього процесу.

Матеріал посібника може використовуватись як основа для лекційного курсу та циклу практичних і лабораторних робіт, що виконуються в освітньому процесі за будь-яких форм його організації, а також у процесі самостійного вивчення основ програмування та вебтехнологій.

1. ВСТУП ДО ВЕБТЕХНОЛОГІЙ

Вебтехнології - сукупність методів, принципів та програмних засобів, за допомогою яких здійснюється введення і виведення інформації у вигляді вебсторінок.

Під вебтехнологіям зазвичай розуміють мови розмітки, мови програмування, системи керування вмістом сайтів та інші технології, які дозволяють створювати вебсайти, програми та магазини.

Є два типи вебтехнологій.

Перший, так званий ***front-end*** - той, що відповідає за зовнішній вигляд сайту, і видимий у браузері користувача. Такі технології включають HTML (HyperText MarkUp Language), CSS (Cascading Style Sheets), JavaScript.

Другий тип вебтехнологій – ***back-end***, який працює на сервері та використовується для обробки даних. Їхня дія зазвичай невидима для користувача, видно лише введені дані або дії, виконані на вебсайті, і результат цих даних або дій. Таким чином, весь процес виконання виходить за межі браузера.

HTML, CSS, JavaScript - це три кити, на яких працюють вебзастосунки.

- HTML - каркас,
- CSS - приємне візуальне оформлення (в CSS3 з’явилась можливість реалізовувати й анімації),
- JavaScript - логіка, інтерактивність та взаємодія з користувачем.

У зовнішньому інтерфейсі практично завжди використовуються HTML, CSS та JavaScript, у внутрішньому інтерфейсі набагато більше свободи вибору.

Популярні мови програмування: PHP, яка була створена спеціально для використання на вебсайтах, більш універсальні мови – Python, Java, C#, C++.

Серверна частина також включає бази даних.

Використовуються СУБД, такі як MySQL або MariaDB. Для опрацювання даних використовується мова запитів SQL.

Для полегшення управління даними були створені спеціальні інструменти, які дозволяють подавати дані у наочній графічній формі, наприклад, phpMyAdmin.

Наприклад, людині потрібно увійти на якийсь форум. Він вводить ім'я та пароль.

Дані, надіслані на сервер, перевіряються. Скрипт, написаний будь-якою з внутрішніх мов програмування, запитує базу даних, чи існує такий користувач і чи правильний пароль.

Якщо це так – скрипт дозволяє користувачеві увійти до системи.

Якщо ні – відобразиться інформація про те, що ім'я або пароль неправильні.

Сценарій та база даних невидимі для користувача - інакше він може вкрасти імена та паролі інших користувачів, які створили облікові записи на форумі.

Системи керування контентом використовуються для полегшення створення, оновлення та публікації контенту на вебсайті. Завдяки простоті використання та широким функціям, вони значно полегшують роботу сайту.

Матеріали для самоосвіти:

https://hostiq.ua/wiki/ukr/cms/

Що таке CMS сайту

Як працює Веб

Клієнти та сервери

Комп'ютери, підключені до мережі, називаються клієнтами та серверами. Спрощена схема їхньої взаємодії може виглядати так (Рис. 1.1):

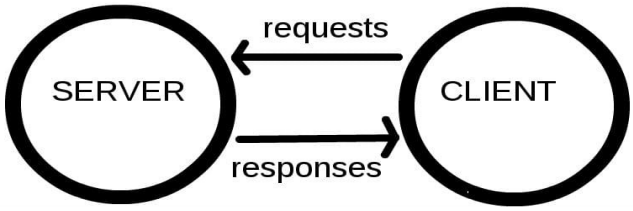


Рис. 1.1. Схема взаємодії клієнта із сервером

Клієнти є звичайними користувачами, підключеними до Інтернету за допомогою пристроїв (наприклад, комп'ютер підключений до Wi-Fi, або телефон підключений до мобільної мережі) та програмного забезпечення, доступного на цих пристроях (як правило, браузер, наприклад Firefox або Chrome).

Сервери - це комп'ютери, які зберігають вебсторінки, сайти або програми. Коли клієнтський пристрій намагається отримати доступ до вебсторінки, копія сторінки завантажується з сервера на клієнтський комп'ютер для відображення у браузері користувача.

Поняття, важливі в контексті взаємодії клієнта та сервера:

- *Підключення до Інтернету*: Дозволяє надсилати та приймати дані через мережу.
- *TCP/IP* (Trasmission Control Protocol / Internet Protocol): Протокол Керування Передаванням та Інтернет Протокол є протоколами комунікацій, які визначають, яким чином дані повинні передаватися мережею.
- *DNS* (Domain Name System) система доменних імен: система перетворення імені хоста (комп'ютера або іншого мережевого пристрою) в IP-адресу. Коли користувач вводить вебадресу у своєму браузері, браузер звертається до DNS, щоб знайти реальну адресу вебсайту, перш ніж він зможе отримати доступ до нього.

- *HTTP* (HyperText Trasfer Protocol): Протокол Передавання Гіпертексту - це протокол, який визначає мову для клієнтів і серверів, щоб спілкуватися один з одним.
- *Файли компонентів:* сайт складається з декількох різних файлів. Ці файли бувають двох основних типів:
  - *Файли коду*: документи HTML, таблиці стилів CSS, файли JavaScript тощо.
  - *Матеріали*: зображення, музика, відео, текстові документи.

Коли користувач вводить вебадресу до свого браузера:

1. Браузер звертається до DNS сервера і знаходить реальну адресу сервера, на якому "міститься" сайт.
2. Браузер надсилає HTTP запит до сервера, просячи його відправити копію сайту для клієнта. Це повідомлення та решта даних, що передаються між клієнтом і сервером, передаються по інтернет-з'єднанню з використанням протоколу TCP/IP.
3. Якщо сервер схвалює запит клієнта, сервер надсилає клієнту статус "200 ОК", який означає, що запит виконано успішно, а потім починає надсилання файлів сайту до браузера у вигляді невеликих порцій, які називають пакетними даними.
4. Браузер збирає пакетні дані у повноцінний сайт і показує його користувачеві.

Вебсервер

Термін вебсервер може стосуватися апаратного чи програмного забезпечення, або обох, що працюють разом.

З боку *апаратного забезпечення* вебсервер - це комп’ютер, на якому зберігається програмне забезпечення вебсервера та файли компонентів вебсайту (наприклад, документи HTML, зображення, таблиці стилів CSS і файли JavaScript). Вебсервер підключається до Інтернету та підтримує фізичний обмін даними з іншими пристроями, підключеними до Інтернету.

З боку *програмного забезпечення* вебсервер включає кілька частин, що контролюють, як вебкористувачі отримують доступ до розміщених файлів. Як мінімум, це HTTP-сервер. Сервер HTTP - це програмне забезпечення, яке розуміє URL-адреси (вебадреси) і HTTP (протокол, який використовує браузер для перегляду вебсторінок). Доступ до сервера HTTP можна отримати через доменні імена вебсайтів, які він зберігає, і він доставляє вміст цих розміщених вебсайтів на пристрій кінцевого користувача.

На базовому рівні, коли браузеру потрібен файл, який розміщено на вебсервері, браузер запитує файл через HTTP. Коли запит досягає правильного (апаратного) вебсервера, (програмний) HTTP-сервер приймає запит, знаходить запитуваний документ і надсилає його назад у браузер, також через HTTP (Рис.

1.2). Якщо сервер не знаходить потрібний документ, він замість цього повертає відповідь 404.

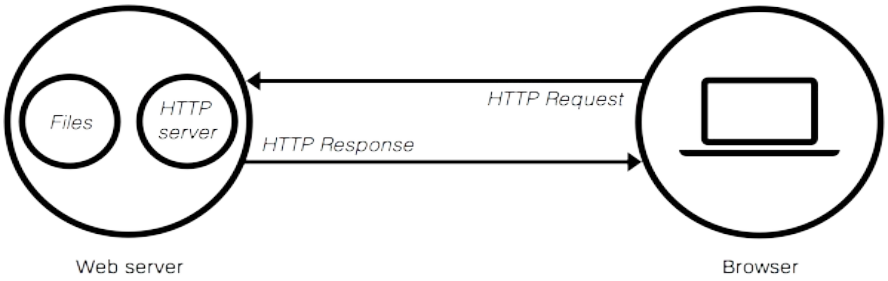


Рис. 1.2. Базове представлення з'єднання клієнт/сервер через HTTP

Щоб опублікувати вебсайт, потрібен статичний або динамічний вебсервер.

Статичний вебсервер складається з комп'ютера (апаратного забезпечення) з сервером HTTP (програмне забезпечення). Він називається «статичним», тому що надсилає розміщені на ньому файли до браузера "як вони є".

Динамічний вебсервер складається зі статичного вебсервера та додаткового програмного забезпечення, як правило, сервера додатків і бази даних. Він називається «динамічним», оскільки сервер додатків оновлює розміщені файли перед тим, як надсилати вміст у браузер через сервер HTTP.

Наприклад, щоб створити остаточні вебсторінки, які користувач бачить у браузері, сервер додатків може заповнити шаблон HTML вмістом із бази даних. Такі сайти, як Wikipedia, містять тисячі вебсторінок. Як правило, вони складаються лише з кількох HTML-шаблонів і гігантської бази даних, а не з тисяч статичних HTML-документів. Це налаштування полегшує підтримку та доставлення вмісту.

Хостинг файлів

Вебсервер має зберігати файли вебсайту, а саме всі HTML-документи та пов'язані з ними ресурси, включаючи зображення, таблиці стилів CSS, файли JavaScript, шрифти та відео.

Технічно користувач може розмістити всі ці файли на своєму комп'ютері, але набагато зручніше зберігати всі файли на спеціальному вебсервері, тому що:

- Виділений вебсервер, як правило, більш доступний (працює).
- За винятком простоїв і проблем з системою, виділений вебсервер завжди підключений до Інтернету.
- Виділений вебсервер може мати постійну (виділену) IP-адресу. (Не всі провайдери надають фіксовану IP-адресу для домашніх ліній.)
- Виділений вебсервер зазвичай обслуговується третьою стороною.

З усіх цих причин пошук хорошого хостинг-провайдера є ключовою частиною створення вебсайту.

2. ВСТУП ДО JAVASCRIPT. ВИКОНАННЯ КОДУ JAVASCRIPT

Основні характеристики мови

- *JavaScript - це високорівнева мова програмування для широкого кола застосувань.*
- *JavaScript - мультипарадигмальна мова, що підтримує об'єктно-орієнтований, подійно-орієнтований, функціональний та імперативний підходи.*
- *JavaScript застосовується для запису послідовних операцій - «сценаріїв» чи «скриптів», її ще називають скриптовою мовою.*
- *JavaScript - динамічна мова, тобто дозволяє визначати типи даних і здійснювати синтаксичний аналіз та компіляцію «на льоту» (Just-in-time compilation (JIT), безпосередньо на етапі виконання програми.*
- *JavaScript - однопотокова мова, тобто може виконувати лише одну операцію одночасно. Але за рахунок механізмів середовища виконання JavaScript здатна працювати асинхронно.*
- *JavaScript – "безпечна" мова програмування. Вона не надає низькорівневого доступу до пам'яті чи процесора, оскільки початково була створена для браузерів, які цього не потребують.*

Найчастіше JavaScript застосовується для програмування функціональності компонентів вебсторінок, створення їх інтерактивності.

Кожен вебзастосунок чи сайт побудований з використанням трьох технологій - HTML, CSS та JavaScript. JavaScript відповідає за інтерактивність вебсторінок і взаємодію з користувачем.

Коли користувач завантажує сторінку в браузері, то спочатку обробляються HTML і CSS, а після цього - скрипти.

JavaScript обробляється в вебдодатках на стороні клієнта, тобто у браузері. Завдяки цьому код може виконуватися у будь-якій операційній системі, а вебінтерфейси, які працюють на його основі, є кросплатформними.

Дедалі частіше компанії не обмежуються роботою із JavaScript лише в браузері.

У 2009 році було створено середовище Node.js з метою виконання коду JavaScript без браузера. Так стало можливим програмістам створювати повноцінні (інтерфейсні та внутрішні) (full-stack - front-end і back-end) програми, використовуючи лише мову JavaScript. Прикладами проєктів за такої комбінації є Netflix та PayPal.

Ядро JavaScript включає цілу низку функцій, що дають наступні можливості:

- Зберігати дані в змінних;
- Активувати частину коду у відповідності з певними сценаріями, які здійснюються на сторінці сайту;
- Створювати контент, який оновлюється автоматично;

- Управляти мультимедійними можливостями (працювати з відео, анімувати зображення).

Ще більше можливостей дає функціонал, який доступний як надбудова щодо основних складових JavaScript - інтерфейси прикладного програмування (Application Programming Interface, API), які істотно розширюють набір інструментів розробника.

Мова не містить деяких корисних інструментів, зокрема стандартної бібліотеки та стандартних інтерфейсів для роботи з серверами і базами даних.

Застосування JavaScript

***JavaScript має широке застосування в таких областях**:*

- *AJAX (Asychronous Javascript and XML - асинхронний JavaScript і XML)* - підхід до побудови користувацького інтерфейсу вебзастосунку, коли вебсторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані, тобто асинхронний обмін даними між браузером користувача та ресурсами сервера.
- *Comet* – спосіб роботи вебдодатків, коли під час HTTP-з'єднання сервер відправляє дані браузеру без додаткових запитів.
- *Браузерні ОС* – код деяких браузерних операційних систем, який переважно (іноді на більш ніж 75%) складається зі скриптів.
- *Закладки (букмарклети)* – JavaScript має широке застосування в роботі програм, що розміщуються в закладках браузера.
- *Браузерні скрипти* – програмні модулі, які виконуються в браузері користувача під час завантаження будь-якого вебдокумента. Приклади сценаріїв скриптів користувача: додавання елементів, автоматичне заповнення форм, відображення вмісту, форматування сторінки, приховування або показ вмісту, додавання інтерактивних елементів на сторінках.
- *Серверні додатки* – JavaScript програми часто виконуються на серверах, які написані іншими мовами. Побудувати серверну логіку без JavaScript, часто, складно (якщо від нього залежать інші компоненти). Крім того, деякі серверні програми використовують спеціальні інтерпретатори, які також не змогли б працювати без JS.
- *Мобільні додатки* – незважаючи на те, що для мобільної розробки ця мова використовується рідко, такі випадки існують.
- *Елементи графічних інтерфейсів.* Наприклад, віджетів. Програмування функціональності віджетів за допомогою JavaScript використовують Google, Apple, Yahoo та «Майкрософт».
- *Прикладне ПЗ.* Наприклад, Google Chrome, вільне середовище робочого столу Gnome та Mozilla Firefox на рушії Quantum працюють на JavaScript.
- *Офісні програми*. Такі програми, як OpenOffice або Microsoft Office, не можна уявити без JavaScript. Там ця мова використовується для створення

макросів, налаштування доступу до вебслужб, інтерпретування будь-яких об'єктів.

- *Освіта.* Ця мова використовується для поглибленого вивчення інформатики або як універсальної першої мови програмування.

Історія створення JavaScript

У 1995 році компанія Netscape Communications прокладала собі стежку у сфері вебтехнологій й відвойовувала позиції у першого браузера NCSA Mosaic. Веб потребував легкої мови, якою просто програмувати. Так з'явилася ідея скриптової мови Mocha, з якою можна працювати в браузері.

Водночас Sun Microsystems завершувала роботу над своєю мовою програмування Java. І Netscape були готові інтегрувати їхню мову у свій браузер. Але Java призначалася для більш обізнаної в програмуванні аудиторії. Це суперечило призначенню Mocha, що мала стати провідником між Java і користувачами, які потребували її для розробки вебсайтів.

Після доопрацювання прототип Mocha почали використовувати в Netscape Communicator й він отримав назву LiveScript. А у грудні 1995 року угода між Netscape Communications і Sun була закрита. Так народилася мова JavaScript, а Java служила для створення складних компонентів у браузері.

У 2015-му нова версія мови, ES6, дала JavaScript друге життя - з'явилися нові стандарти, можливість працювати з константами та виконувати багато функцій при скороченому коді. Цей реліз - єдиний, що підтримується всіма браузерами, хоча є й покращені версії.

У 2020 році JavaScript вперше випередила Java, і стала найпопулярнішою мовою програмування. Згідно з рейтингом DOU, у 2022 році цією мовою писали 18,8% розробників, і вона залишається на першому місці.

Мови "над" JavaScript

Синтаксис JavaScript не задовольняє потреби всіх розробників і всіх проєктів. Існують різні нові мови, які *транспілюються* (конвертуються) в JavaScript до того, як виконаються в браузері.

Сучасні інструменти роблять транспіляцію дуже швидкою і прозорою, дозволяючи розробникам писати код іншою мовою і автоматично конвертувати його в JavaScript.

Приклади таких мов:

- CoffeeScript - вводить більш короткий синтаксис, ніж у в JavaScript, що дозволяє писати більш чіткий і точний код. Зазвичай, це до вподоби програмістам на Ruby.
- TypeScript - додає "строгу типізацію даних", щоб спростити розробку і підтримку складних систем. Розробляється у Microsoft.
- Flow - також додає типізацію даних, але іншим способом. Розробляється компанією Facebook.

- Dart - це автономна мова, яка має власний рушій, що працює в не-браузерних середовищах (мобільні застосунки), але також може транспілюватися в JavaScript. Розробляється компанією Google.
- Brython - це транспілятор коду з мови Python в JavaScript, що дозволяє писати застосунки на чистому Python без використання JavaScript.
- Kotlin - це сучасна, лаконічна і безпечна мова програмування, яку можна компілювати для браузера або NodeJS.

Існують й інші мови. При використанні транспілювальних мов слід знати JavaScript для повного розуміння процесу розробки.

Виконання коду JavaScript. Рушій і середовище виконання JavaScript

Для виконання коду Javascript використовується програмне забезпечення, яке називається ***рушієм JavaScript (JavaScript Engine).***

Рушій, у свою чергу, вбудований в програмне середовище, яке називається ***середовищем виконання JavaScript (JavaScript Runtime Environment)***.

Приклади середовищ виконання: браузери, Node.js, Java Runtime Environment (JRE).

Середовище значною мірою визначає функціонал JavaScript. Наприклад, Node.js підтримує функції, що дозволяють JavaScript читати/записувати довільні файли, здійснювати мережеві запити тощо.

У браузері JavaScript може виконувати задачі, пов'язані з маніпуляцією вебсторінками, взаємодією з користувачем та вебсервером.

Браузер має вбудований рушій, який ще називають "віртуальною машиною JavaScript".

Наприклад:

V8 – в Chrome, Opera та Edge.

Gesko – в Firefox.

"Chakra" - в IE,

"JavaScriptCore", "Nitro" і "SquirrelFish" - в Safari, та інші.

Середовище виконання JavaScript - це контейнер, що містить усе потрібне для використання JavaScript. Серцем будь-якого середовища виконання JavaScript є рушій JavaScript.

У контексті браузера середовище виконання JavaScript складається з таких елементів:

1. Рушій JavaScript.
2. Вебінтерфейси API.
3. Черга зворотного виклику.
4. Цикл подій.

Рушій JavaScript

Рушій (engine) JavaScript - це програма, головним завданням якої є читання та виконання коду. Рушії JavaScript вбудовані в усі сучасні браузери.
Ця програма працює так:

- читає код;
- перекладає його в машинний код;
- запускає цей машинний код.

Рушій JavaScript складається з 6 основних компонентів, які працюють разом як одне ціле (Рис.2.1).

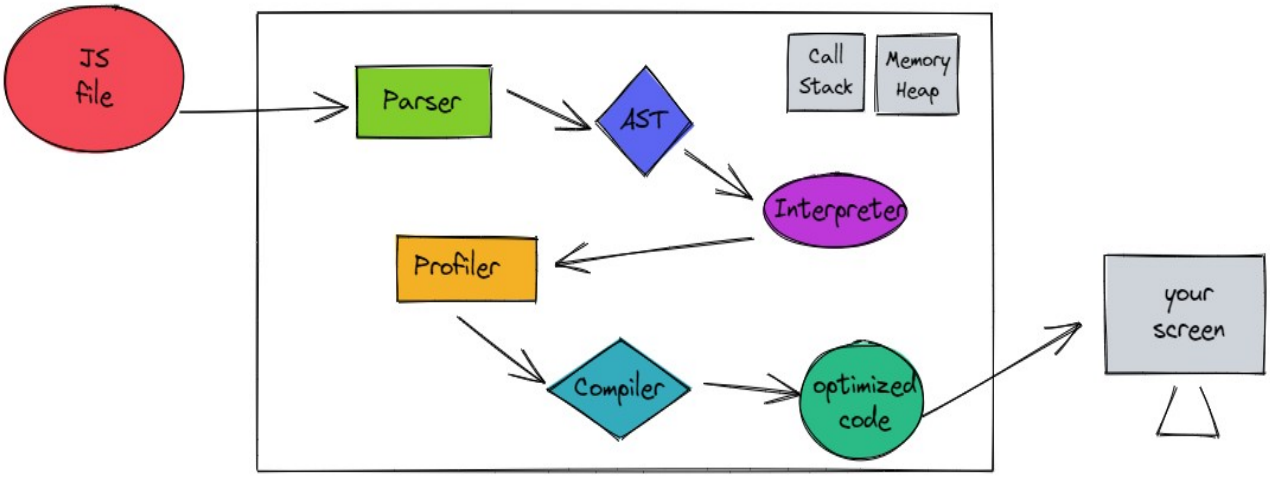


Рис. 2.1. Компоненти рушія JavaScript.

1. Синтаксичний аналізатор (parser)

Вихідний код надсилається до декодера, який здійснює лексичний аналіз коду і декодує вміст усередині тегу сценарію HTML у токени, які надсилаються до парсера. Рушій намагається уникати аналізу непотрібного коду відразу, щоб заощадити час.

Лексичний аналіз є найпершим кроком компіляції або інтерпретації. Він також називається токенізацією. В процесі лексичного аналізу послідовність символів перетворюється у послідовність лексичних елементів (токенів). Токен повинен мати ім’я та може мати значення.

Токен можна символічно визначити як
`<name, value(optional)> (<ім’я, значення (необов’язково)>).`
Наприклад, є код:

```
var x = 1;
```

Токенізатор розіб'є його на п'ять токенів -

```
<KEYWORD, var> <ID, x> <EQUALS> <INTEGER, 1> <SEMICOLON>
```

Аналізатор синтаксису (парсер) читає токени та аналізує відповідно до правил мови.

Наприклад, правило мови (визначає присвоєння):

```
assignment = ID followed-by EQUALS followed-by INTEGER
```

Відповідно до цього правила, x = 1 є присвоєнням. Але var x = 1 або x = y або x = 2.3 згідно з цим правилом не є присвоєннями. Хоча ми знаємо, що це теж присвоєння. Визначення правил і аналіз цих правил є складною роботою.

2. Абстрактне синтаксичне дерево. (Abstract Syntax Tree, AST)

На основі отриманих токенів аналізатор створює вузли. За допомогою цих вузлів він створює абстрактне синтаксичне дерево.

Якщо токен-ланцюжок відповідає синтаксису мови, аналізатор синтаксису генерує абстрактне синтаксичне дерево (AST), інакше видає помилку.

AST - це деревоподібне подання вихідного коду.

Наприклад, `x = 1` буде подано у вигляді простого абстрактного синтаксичного дерева (Рис. 2.2):

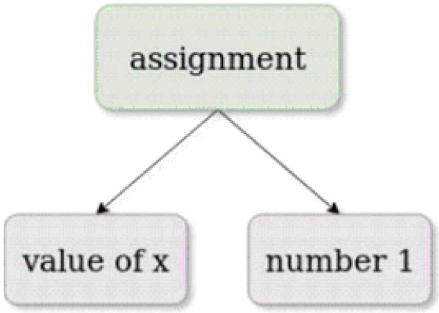


Рис. 2.2. AST для виразу `x = 1`.

Приклад більш складного AST.

Нехай є така функція

```
function square (x) {var result = x * x; return result;}
```

Подання AST виглядатиме так (Рис. 2.3):

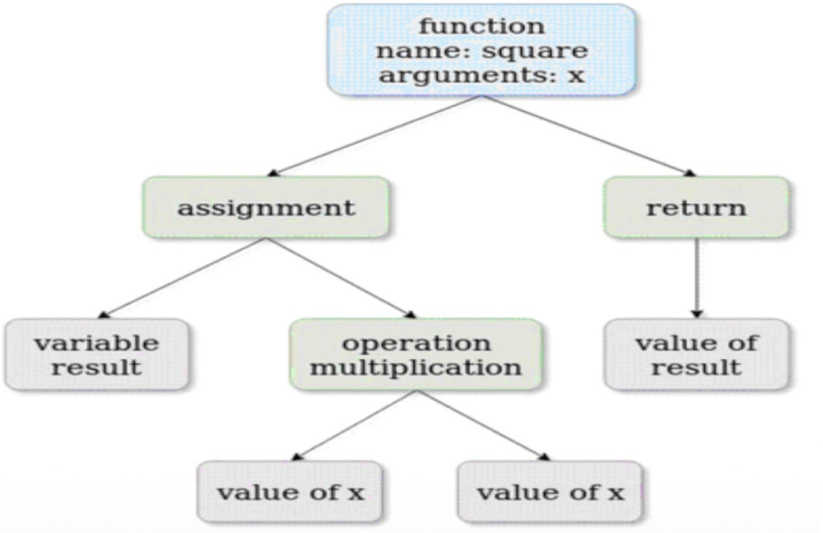


Рис. 2.3. Приклад AST.

3. Інтерпретатор - Коли AST готове, компілятор/інтерпретатор генерує з нього байт-код, зчитуючи код рядок за рядком. Після створення байт-коду AST видаляється, а простір пам’яті звільняється.
4. Профайлер - відстежує та спостерігає за кодом для його оптимізації.
5. Компілятор – працює на випередження та створює переклад написаного коду та компілює його до мови нижчого рівня, яку можуть читати машини.
6. Оптимізований код. Код потрібно оптимізувати, щоб працювати швидше.

Компіляція чи інтерпретація – це складна робота. Якщо фрагмент коду потрібно інтерпретувати кілька разів, це цілком нормально. Якщо якийсь код потрібно інтерпретувати багато разів (він міститься у довготривалому циклі або код у часто викликаній функції), тоді інтерпретувати весь час буде неефективно. Щоб вирішити цю проблему, рушії JavaScript використовують спостерігач для підрахунку використання кожної частини коду. На основі звіту спостерігача рушій вирішує компілювати якийсь код, а іноді й оптимізувати.
Для виконання компіляції рушії JavaScript використовують компілятор *Just In Time (JIT).*

Стек викликів і купа пам'яті

Стек викликів і купа пам’яті є важливими частинами рушія JS.

Стек викликів (Call Stack) - механізм, що допомагає інтерпретатору JavaScript відстежувати виконання функцій, які викликає скрипт.

Стек викликів діє за принципом LIFO (останній зайшов, перший вийшов), тобто першою завжди обробляється функція на верхівці стеку.

Відомо, що *JavaScript однопотокова мова. Це означає, що JavaScript може виконувати лише одну операцію за раз і має лише один стек викликів.*

Купа пам’яті (Memory Heap) - це місце, де відбувається розподіл пам'яті, необхідної для виконання програми. Це неструктурований пул пам'яті, в якому тимчасово зберігаються всі компоненти, які потрібні для застосування.

Стек викликів викликає функцію з купи пам’яті. Щоразу, коли функція викликається, вона з'являється у стеку викликів. Коли ж функція завершує виконання, інтерпретатор прибирає її звідти.

Функція припиняє діяти, якщо повернулось значення з return або ж коли всі її інструкції виконано.

Коли функція починає викликати інші функції, стек викликів наповнюється записами.

Коли досягнуто максимального стека викликів, наприклад, з нескінченним циклом, це називається переповненням стеку.

Максимальний розмір стеку викликів сягає від 10 до 50 тисяч викликів. Якщо у програмі стек переповнився, то, ймовірно, у цьому винен нескінченний цикл.

Браузер захищає вебсторінку від підвисання, обмежуючи стек викликів.

Середовище виконання JavaScript

Це - середовище, в якому виконується код. Система середовища виконання полегшує зберігання функцій, змінних і керування пам’яттю за допомогою структур даних, таких як черги, купи та стеки.

JavaScript Runtime - це програмне забезпечення, яке розширює функціонал рушія JavaScript. Воно надає функції/інтерфейси API для створення програмного забезпечення на основі Javascript.

І браузери, і фреймворки на основі JavaScript мають середовища виконання, які є різними залежно від їхніх потреб.

Основні складові JavaScript Runtime:

1. Вебінтерфейси API.
2. Черга зворотного виклику.
3. Цикл подій.

1. ***Веб інтерфейси прикладного програмування***
Web API (Application Programming Interface)

Інтерфейси прикладного програмування (API) є надбудовою до основних складових JavaScript і істотно розширюють набір інструментів розробника.

Вебінтерфейси API не є частиною механізму JavaScript, але вони є частиною середовища виконання.

Браузер або будь-яке інше середовище виконання надає API для зв’язку із зовнішнім світом.

API написані низькорівневими мовами програмування (на зразок C). Саме тому їхні можливості дуже відрізняються від стандартної мови JavaScript.

В основному надані API визначають функціональні можливості платформи. Браузер надає API, придатні для запуску програми в браузері. Node.js надає API, застосовні для серверної програми.

API – це готові модулі коду, які допомагають програмісту в реалізації різноманітних складних завдань. Зазвичай такі «заготовки» діляться на браузерні і API третіх розробників.

У сучасних браузерах доступна велика кількість API, які дозволяють виконувати різноманітні дії, зокрема:

- ***Маніпулювання документами – DOM API*** (DOM, Document Object Model - об’єктна модель документа) дозволяє розробникам маніпулювати HTML і CSS, щоб створювати, змінювати і видаляти HTML і динамічно застосовувати стилі до вебсторінок.
- ***Малювання графіки та керування нею***. Canvas API і API бібліотеки вебграфіки (Web Graphics Library, WebGL) дозволяють програмно оновлювати піксельні дані, що містяться в елементі <canvas>.

- ***Отримання даних із сервера.*** Fetch API надає інтерфейс для отримання ресурсів у мережі за допомогою загальних визначень об’єктів Request і Response.

До сторонніх інтерфейсів належать, наприклад, API соціальних мереж Twitter і Facebook.

Наприклад, відомо, що в JavaScript можна виконати лише одну операцію за раз. А в браузері можна виконувати деякі задачі паралельно (завдяки його API).

Якби код виконувався в інтерпретаторі JavaScript, то не було б можливості виконувати інші операції, поки не прийде відповідь із сервера. Тоді вебзастосунки стали б надзвичайно повільними. Тож браузери пропонують спеціальні API, виконання яких обробляється безпосередньо платформою, тому не блокує стек викликів JavaScript.

2. Черга зворотного виклику(Callback queue)

У черзі зворотного виклику зберігаються функції зворотного виклику, надіслані з вебінтерфейсів API, у порядку їх додавання. Ця черга є структурою даних, яка працює в порядку FIFO (First In - First Out).

Функції зворотного виклику залишатимуться в черзі, доки стек викликів не буде порожнім, потім вони переміщаються в стек циклом подій.

Функція зворотного виклику або callback-функція – це функція (A), що передається як аргумент до іншої функції (B), котра має виконати аргумент A у певний момент часу (викликати колбек-функцію A у відповідь – call back).

3. Цикл подій (Event loop)

Цикл подій постійно контролює стан стеку викликів і чергу зворотних викликів. Якщо стек порожній, він захопить зворотний виклик із черги зворотного виклику та помістить його в стек викликів, запланувавши його для виконання.

Надсилаючи зворотні виклики з вебінтерфейсів API до черги зворотних викликів, цикл подій може постійно додавати ці зворотні виклики до стеку викликів, що створює ілюзію асинхронної роботи JavaScript.

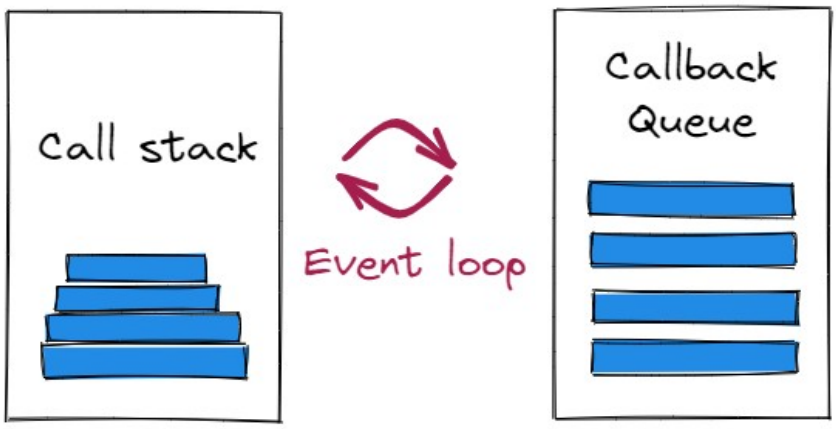


Рис. 2.4. Схема роботи черги зворотного виклику і циклу подій.

Спільна робота JavaScript Engine і Runtime

Перше, що має відбутися перед виконанням коду, - щоб JavaScript зрозумів код. Це означає аналіз коду на відповідність граматиці мови JavaScript і визначення формату речення.

Після розуміння коду JavaScript має виконувати деякі інші завдання для виконання коду: вирішення функцій, значень параметрів, керування поверненнями результатів функцій, упорядкування викликів функцій, збір сміття у купі пам’яті та підготовка машинних інструкцій. Все це робота рушія.
Після того, як рушій завершить свою роботу, починає працювати середовище виконання. Воно перевіряє чергу зворотних викликів, а цикл подій захоплює все, що всередині нього, щоб запланувати його виконання.

3. ІНТЕГРАЦІЯ JAVASCRIPT-КОДУ В HTML-СТОРІНКУ

Вебдокумент може містити не тільки теги (дескриптори), що визначають зовнішній вигляд документа при відображенні його у браузері, але й особливий *програмний код JavaScript - сценарій.*

Сценарій розташовується в спеціальному блоці. Блок виділяється тегами <script> та </script>.

Блоків зі сценаріями у вебдокументі може бути декілька, і вони можуть знаходитись у різних місцях документа.

Наприклад, блок зі сценарієм дозволяється розміщувати в блоці заголовку документа (між <head> та </head>) або блоці основного тіла документа (між <body> та </body>).

Місце розташування блоку зі сценарієм має значення: сценарій виконується при завантаженні документа.

Є кілька способів додати сценарій JavaScript до відповідної HTML-сторінки.

1. Розміщення коду JavaScript всередині пари HTML-тегів <script></script>

Код JavaScript можна вставити у будь-яку частину HTML документа (Рис. 3.1, 3.2).

Приклад.

```
<html>
  <head>
    <script>
      alert("JavaScript code");
    </script>
  </head>
  <body>
  </body>
</html>
```

Зауваження. alert(повідомлення); - одна з функцій JavaScript для взаємодії з користувачем. Вона показує модальне вікно з повідомленням. Наприклад, `alert("Привіт, світ!")`

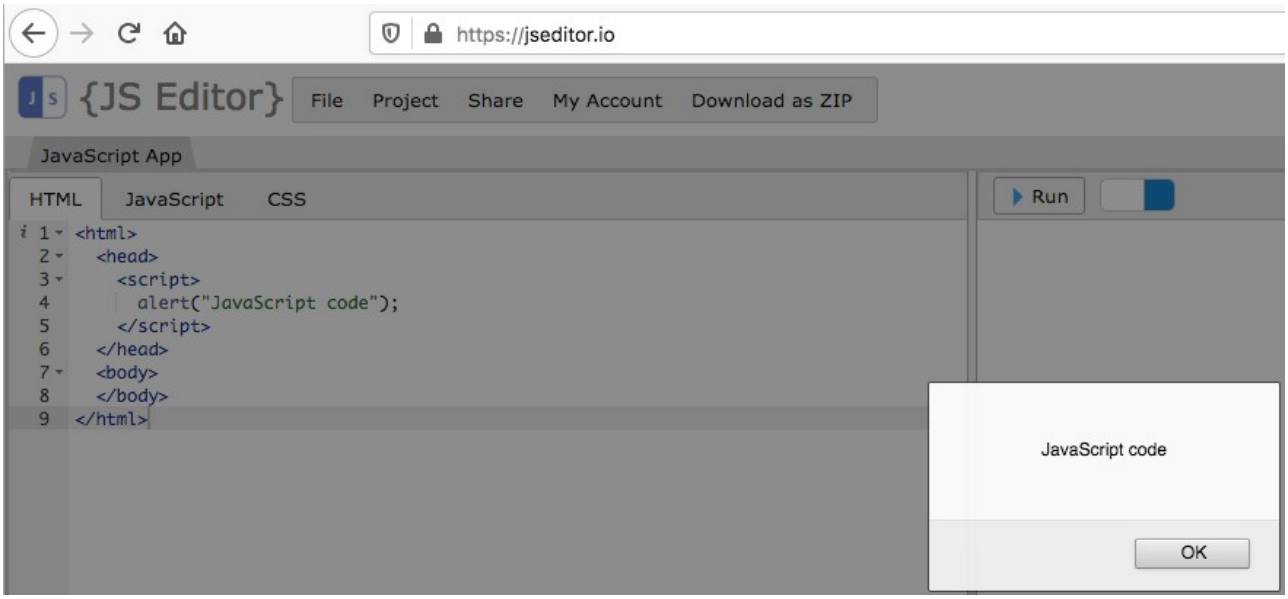


Рис. 3.1. Код JavaScript всередині тегів <script></script> в голові документа.

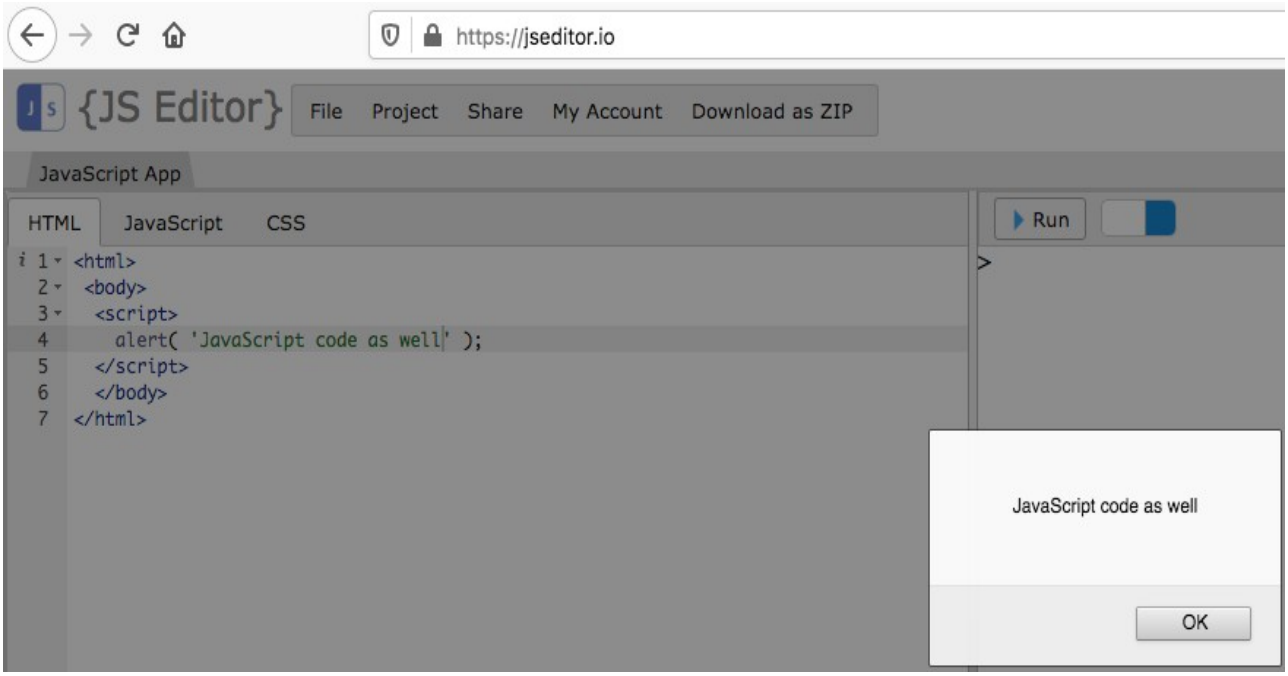


Рис. 3.2. Код JavaScript всередині тегів <script></script> в тілі документа.

Тег <script> має декілька атрибутів, які рідко використовуються, але можуть міститися в старому коді.

1. `type: <script type=…>`

Старий стандарт HTML4, вимагав наявності цього атрибута. Зазвичай його значення було "text/javascript":

```
<script type="text/javascript">
```

В сучасному стандарті HTML5 цей атрибут має інший зміст. Він використовується для модулів JavaScript.

2. `language: <script language=…>`

```
<script language="javascript">
```
Цей атрибут вказував на мову скрипта. Зараз цей атрибут не потрібний, оскільки JavaScript є усталеною мовою.

3. Коментарі до та після скриптів.

У старих посібниках можна знайти коментарі всередині тега `<script>`, наприклад:

```
<script type="text/javascript"><!--
    ...
//--></script>
```

Коментарі ховали код JavaScript для старих браузерів, які не мали засобів опрацювання тегу <script>. Браузери, випущені за останні 15 років, не мають цієї проблеми.

2. Виклик JavaScript-коду як обробника деякої події

В основному виконання коду JavaScript керується подіями: виконання коду є “адекватною” реакцією на дії користувача.

Приклади подій: натиснення кнопки, завантаження сторінки, зміна вмісту текстового поля, перемикання прапорця тощо. З кожним HTML-елементом пов’язаний свій набір подій.

Імена всіх подій мають префікс **"on":** "onClick", "onLoad", "onChange" тощо.

Щоб зробити JavaScript-код обробником деякої події, потрібно у тегу, який описує HTML-елемент, написати атрибут, що є іменем події, і присвоїти йому значення, яке є кодом JavaScript:

```
<... ім’я події = код JavaScript>
```

Приклад. Опис кнопки, при натисненні якої (тобто, при настанні події onClick) з’являється віконце з написом “JavaScript code” (Рис. 3.3):

```
<form><input type="button" value="Example"
onClick="alert('JavaScript code')"></form>
```

На практиці JavaScript-код часто буває занадто об’ємним для прямого розміщення у тегу. Тоді код доцільно оформити у функцію і викликати функцію як обробник події.

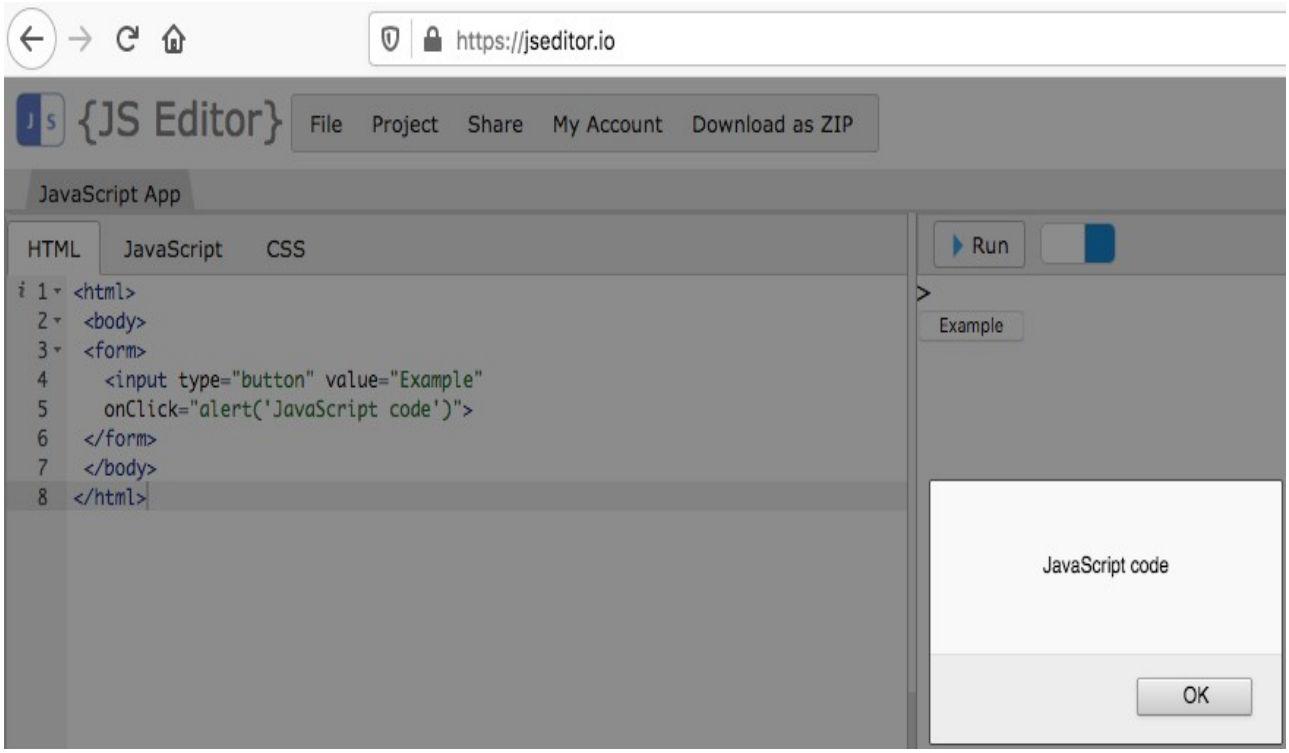


Рис. 3.3. Приклад виконання сценарію.

3. Вставка JavaScript-коду як значення атрибуту гіперпосилання href HTML-елемента <a>

Щоб браузер міг розрізнити URL і код JavaScript, перед JavaScript-кодом слід написати “javascript:”

Приклад. У HTML-коді оголошені два гіперпосилання:

```
<a href="http://www.google.com.ua">Web page</a>
<br>
<a href="javascript: alert('JavaScript code!')"></a>
```

При натисканні першого гіперпосилання відбудеться перехід на сторінку з URL = http://www.google.com.ua (Рис.3.4).

При натисканні на друге гіперпосилання буде виведено віконце з текстом "JavaScript code!". Слід вказати “префікс” “javascript:”, інакше браузер опрацює значення атрибута href як звичайне посилання.

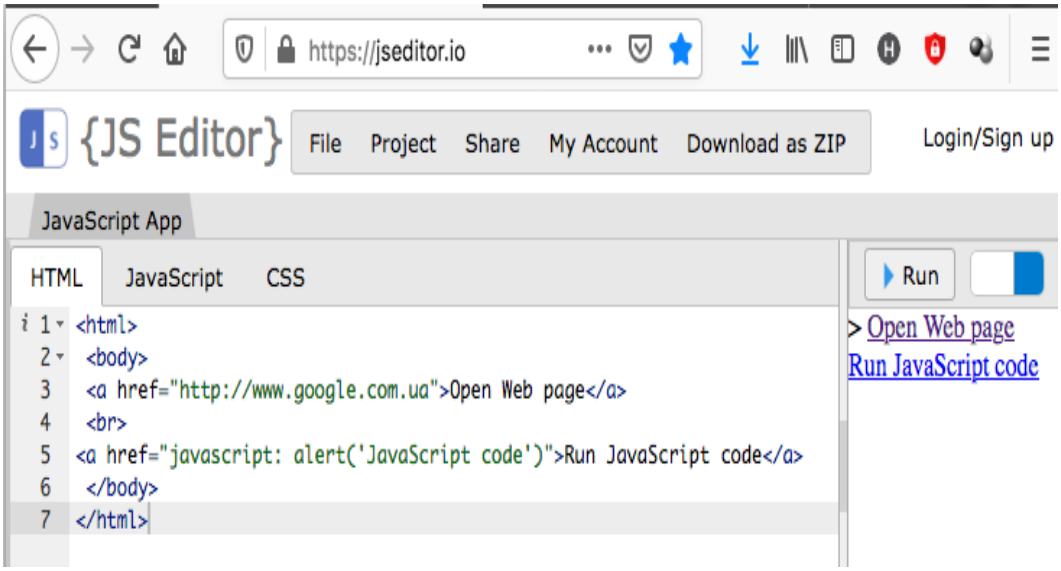


Рис. 3.4. Приклад сценарію.

4. Розміщення JavaScript-коду в окремому зовнішньому файлі із розширенням .js

Як правило, тільки найпростіші скрипти поміщаються безпосередньо в HTML. Більш складні знаходяться в окремих файлах.

Для застосування коду такий файл - зовнішній скрипт - слід підключити до HTML-файлу.

Для підключення використовується атрибут src тегу script:

```
<script src="/шлях_до_файлу_з_розширенням_js"></script>
```

В старих джерелах може бути вказано

```
<script language="javascript" src="/path/to/script.js"></script>
```

Тут /path/to/script.js - абсолютний шлях до файлу скрипта з назвою script.js з кореня сайту.

…

Можна вказати відносний шлях з поточної сторінки:

```
src="script.js"
```

або

`src="./script.js"` - файл "script.js" зберігається у поточній теці.

Можна вказати повну URL-адресу:

```
<script src=
"https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/
lodash.js">
</script>
```

Можна підключати кілька скриптів, використовуючи кілька тегів:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
```

Це найбільш професійний спосіб інтеграції JavaScript в HTML.
Переваги цього способу:

1. HTML-верстальник і JavaScript-програміст можуть працювати одночасно, не заважаючи один одному, оскільки кожен працює над окремим файлом.
2. Браузер завантажує зовнішній файл та зберігає у своєму кеші. Інші сторінки, які посилаються на один і той самий скрипт, замість повторного завантаження файлу, будуть брати його з кешу, тому файл буде завантажено лише один раз. Це зменшує трафік і робить сторінки швидшими.
3. У великих файлах, як правило, фрагменти коду застосовують багаторазово на різних сторінках. Збереження спільних частин коду в окремому місці дозволяє уникнути засмічення коду і зменшує, таким чином, загальний обсяг коду сайту.
4. Код легко редагувати, оскільки весь він компактно, без повторень, розміщений у виділеному місці.

Приклад. В одному каталозі містяться файли onlyHTML.html і onlyJavaScript.js.
Вміст файлу onlyJavaScript.js:

```
function info() {
alert("JavaScript code from file onlyJavaScript.js");
}
```

Вміст файлу onlyHTML.html:

```
<html>
  <body>
    <script src="onlyJavaScript.js"></script>
       <input type="button" id="myButton" value = "Press it!"
onclick="info();">
  </body>
</html>
```

Приклад виконання сценарію наведено на Рис.3.5.

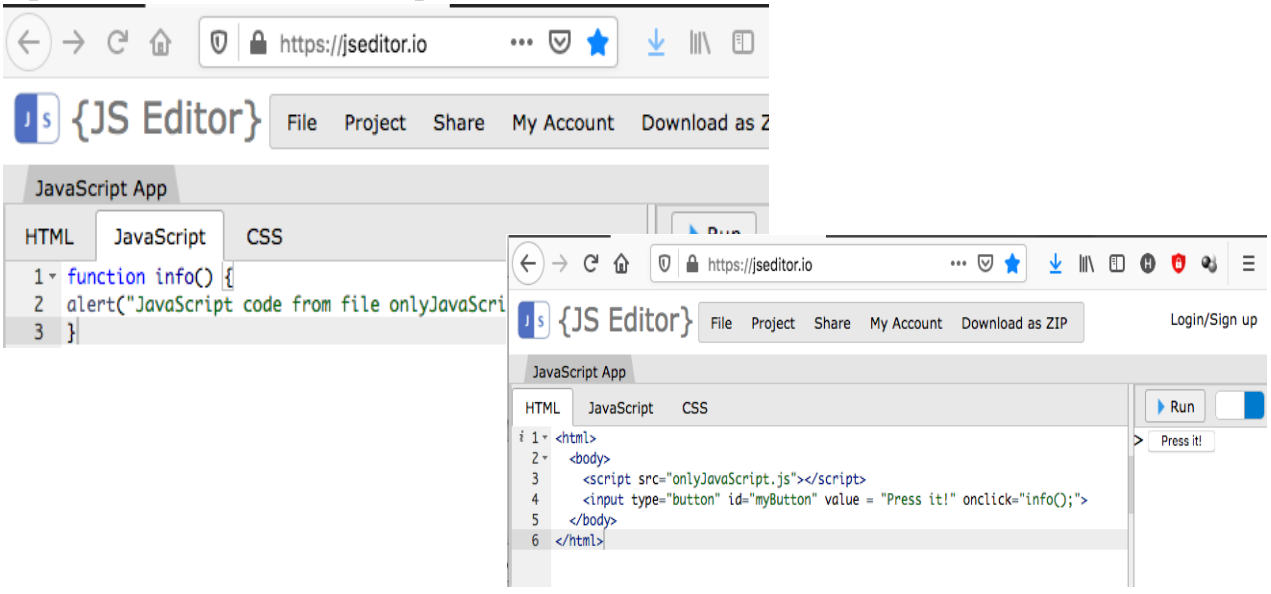


Рис. 3.5. Приклад виконання сценарію.

При натисненні кнопки, оголошеної у файлі onlyHTML.html, виконається функція info(), визначена у файлі onlyJavaScript.js (Рис. 3.6).

Якщо викликається функція, описана в окремому файлі з розширенням .js, а заданий файл не знайдено, то виникне помилка.

У файлі з розширенням .js може міститися виключно JavaScript-код (не HTML-код і не каскадні таблиці стилів).

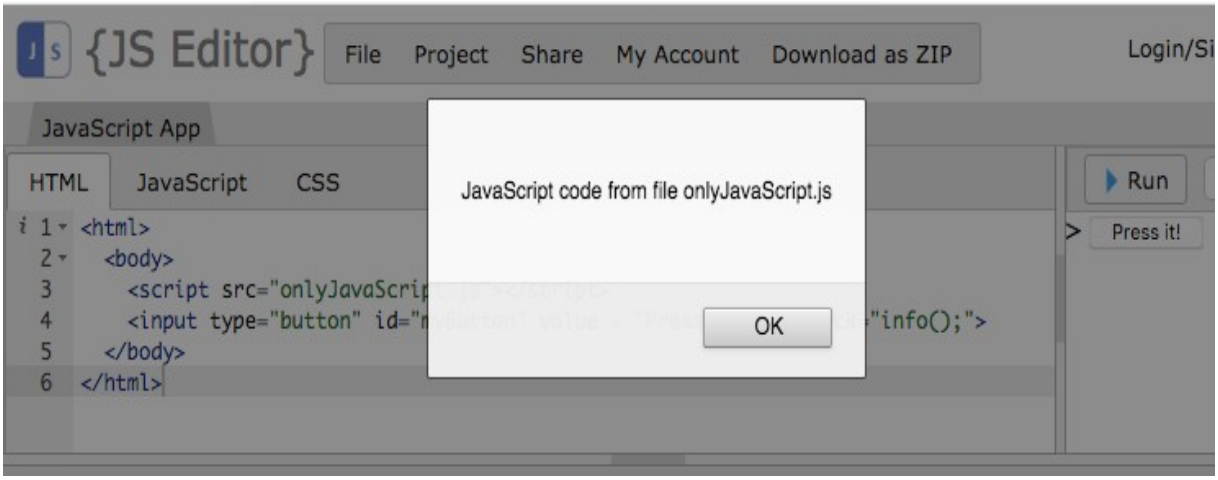


Рис. 3.6. Приклад виконання сценарію.

Сучасний режим, "use strict“

Впродовж тривалого часу JavaScript розвивався без проблем із сумісністю. До мови додавалися нові функції, а стара функціональність залишалася незмінною. Перевагою цього було те, що наявний код не ламався. Проте, будь-яка помилка або неідеальне рішення назавжди ставали частиною JavaScript, тому що цей код не змінювався.

У 2009 році, коли з’явився стандарт ECMAScript 5 (ES5), він додав нові функції до мови і змінив деякі наявні. Щоб старий код лишався робочим, більшість таких модифікацій усталено було вимкнено. Щоб увімкнути цей функціонал, потрібно прописати спеціальну директиву: "use strict".

“use strict”

Директива виглядає як рядок: "use strict" чи 'use strict' (“використовувати суворий (режим)”). Якщо вона написана на початку скрипта, він буде виконуватися у “сучасному” режимі.

Приклад:

```
"use strict";
// цей код працюватиме у сучасному режимі
...
```

"use strict" можна писати на початку функції. Таким чином, суворий режим буде використовуватися лише в межах цієї функції. Проте зазвичай цей режим використовується для всього скрипту. Вище "use strict" можуть бути лише коментарі.

use strict не можна скасувати - немає директиви на зразок "no use strict", яка могла б повернути старий режим.

Налагодження і виконання коду JavaScript

1. Онлайн середовища програмування

JS Editor

https://jseditor.io/

Ideone

https://ideone.com/

Replit

https://replit.com/languages/nodejs

Rextester

https://rextester.com/l/js_online_compiler

JS Bin

https://jsbin.com/?js,output

2. Інструменти розробника у браузері

Контекстне меню порожньої сторінки браузера - Дослідити - Консоль - Запустити.

Більшість браузерів для відкриття консолі розробника використовують клавішу *F12*.

Зазвичай, після введення одного рядка коду в консоль і натиснення Enter, він виконується.

Щоб ввести декілька рядків коду, натискають Shift+Enter. Так можна вводити і виконувати довгі фрагменти JavaScript коду (Рис. 3.7).

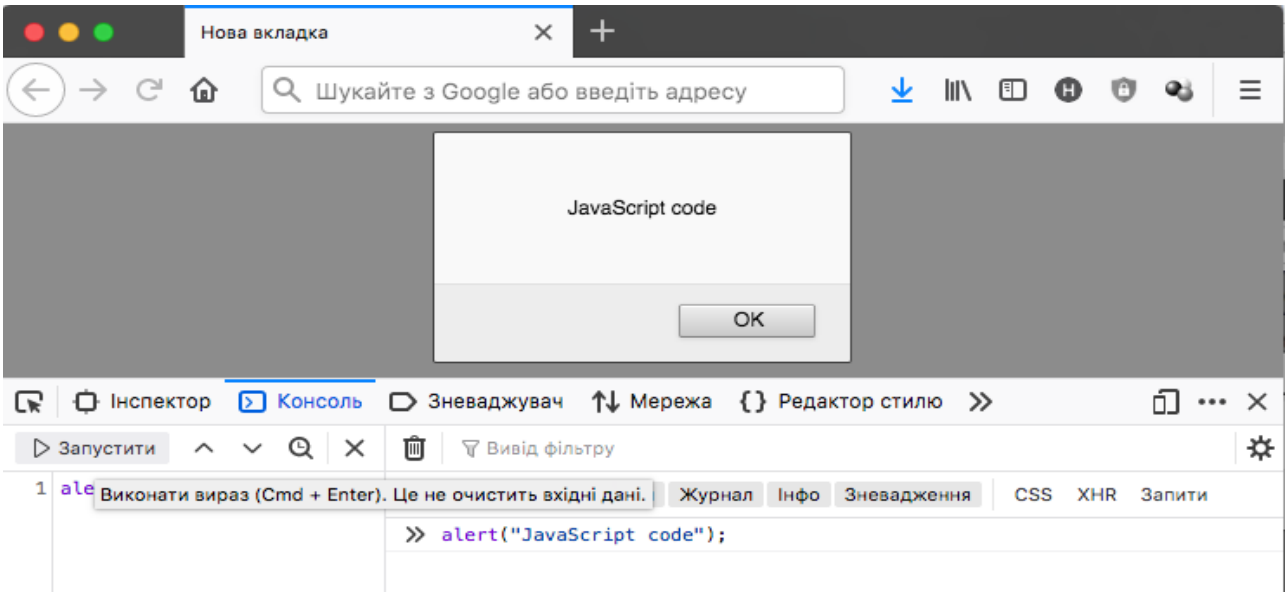


Рис. 3.7. Консоль розробника в браузері.

3. Текстовий редактор і браузер.

Документ одночасно відкривається у браузері та текстовому редакторі (бажано з підсвіченням тегів). Далі в процесі роботи:

1. У текстовому редакторі вносяться зміни у код.
2. Документ зберігається у текстовому редакторі.
3. У браузері виконується команда оновлення сторінки (Reload, Оновити поточну сторінку, Оновити вкладку, Ctlr-R).
4. Зміни документа аналізуються у браузері.

4. Редактори коду (IDE та легкі редактори)

Термін *IDE (Integrated Development Environment - Інтегроване середовище розробки)* означає потужний редактор з багатьма можливостями, що зазвичай працює з цілим програмним проектом, який може мати багато файлів. IDE завантажує проект, дозволяє перемикатися між файлами, надає можливість автозаповнення, яке базується на цілому проекті (не лише на відкритому файлі), інтегрується із системою контролю версій, надає можливість розгортання проекту на тестове середовище та багато інших функцій.

Приклади IDE:

Visual Studio Code (багатоплатформовий, безкоштовний):
https://code.visualstudio.com/

WebStorm (багатоплатформовий, платний):
http://www.jetbrains.com/webstorm/

"***Легкі редактори***" менш потужні, ніж IDE, вони переважно використовуються, для швидкого редагування одного або кількох файлів.

Головна відмінність між IDE та легкими редакторами:

IDE працює на рівні проекту, тому завантажує набагато більше даних під час запуску, і якщо потрібно, аналізує його структуру. Легкий редактор набагато швидший, якщо необхідно відредагували лише один файл.

Легкі редактори можуть мати багато плагінів, включаючи аналізатори синтаксису на рівні проекту, автозаповнення тощо. Це значно розширює їх можливості, тому зараз немає чіткої межі між легкими редакторами та IDE.

Приклади легких редакторів:

Atom (багатоплатформовий, безкоштовний):
https://atom.io/

Sublime Text (багатоплатформовий, безкоштовний на час випробувального терміну):

http://www.sublimetext.com/

Notepad++ (Windows, безкоштовний):
https://notepad-plus-plus.org/

Матеріали для самоосвіти:
https://www.softwaretestinghelp.com/javascript-ide-and-online-code-editors/
https://uk.javascript.info/code-editors

4. ОСНОВИ МОВИ ПРОГРАМУВАННЯ JAVASCRIPT. ТИПИ ДАНИХ. БАЗОВІ СТРУКТУРИ АЛГОРИТМІВ

Мова програмування (англійською programming language) - це система позначень для опису алгоритмів та структур даних.

Будь-яка мова програмування має такі 3 основні складові:

- алфавіт,
- синтаксис (правила написання),
- семантику (правила тлумачення й виконання).

Алфавіт мови JavaScript включає:

- великі та малі літери латиниці;
- арабські цифри;
- пробіл, символи табуляції, ознака кінця рядка тощо;
- спеціальні символи , . ; : ? ‘ ” ! | / \ ~ ( ) [ ] { } < > # % ^ & - + * =

Літери української абетки можна використовувати лише для запису коментарів.

Мова JavaScript *чутлива до регістру*, тобто інтерпретатор розрізняє великі та малі літери.

Програмний код JavaScript складається з інструкцій (команд, вказівок).

Вказівки завершуються крапкою з комою.

Для полегшення читання і розуміння коду рекомендується кожну вказівку писати в окремому рядку.

Приклад:

```
alert('Привіт'); alert('Світ');

alert('Привіт');
alert('Світ');

alert('Привіт')
alert('Світ')
```

Коментар - це частина тексту програми для пояснення програми чи окремих вказівок і не впливає на виконання програми.

Коментарі можуть бути:

- однорядкові, що починаються з //;
- багаторядкові, що беруться в сполучення символів /* і */.

Вкладені коментарі не підтримуються:
не може бути /*...*/ всередині /*...*/.

Приклад.

```
// Цей коментар займає весь рядок
alert('Hello');
alert('World'); // Цей коментар міститься після інструкції

/* Закоментований код
```

```
alert('Hello');
*/
alert('World');
```

Літерал - стале значення певного типу даних, записане у вихідному коді комп'ютерної програми. Це найпростіший (для сприйняття) вид даних, з якими може працювати програма:

- ціле число, наприклад, 15, +5, -174;
- дійсне число, у якому цілу частину відділяють від дробової крапкою. Наприклад, 99.15, -32.45, 2.73e-7 (експоненційна форма запису для числа 2,73 · 10–7);
- логічне значення true (істина) і false (хибність);
- рядок (тексту) - послідовність символів між одинарними або подвійними лапками. Наприклад: "ваше ім'я", 'ваше прізвище'.

Величина - одиниця даних, якими оперує програма. Вона має такі властивості:

- *назва* (ідентифікатор) - послідовність літер латиниці, цифр, знаків нижнього підкреслювання «_» і долара $, перший символ не має бути цифрою;
- *тип* - визначає обсяг відведеної пам'яті, можливі дії, правила тлумачення бітів пам'яті та множину допустимих значень;
- *розмірність* - проста або складена (структурована);
- *значення* - елемент множини допустимих значень величини.

Змінні - вид даних, значення яких дозволено змінювати протягом виконання програми.

Змінні можуть бути *глобальними* або *локальними*. Глобальні змінні досяжні з довільного місця сценарію. Область дії локальних змінних обмежено кодом функції, всередині якого оголошено ці змінні. При створенні сценаріїв JavaScript рекомендовано оголошувати змінні до їхнього використання та надавання початкових значень. Це спрощує налагодження сценаріїв і зменшує ймовірність помилки.

Оголошення змінної:

```
var назва змінної1; // давніший варіант
let назва змінної2;
```

Оголошення константи:

```
const назва константи;
```

Матеріали для самоосвіти
https://learn.matviy.pp.ua/JavaScript/ - про різницю let, var, const

Змінній можна надати значення і при її оголошенні, і далі у коді програми.

Оператор надання значення позначається знаком =, що не є знаком порівняння.
Приклади.

```
let a; // оголошення змінної
a = 3; // надання значення змінній

let a = 3;// оголошення змінної з наданням їй значення

// оголошення трьох змінних з одночасним наданням їм значень
let a = 1;
let b = 2;
let c = 3;

let a = 1, b = 2, c = 3;
// оголошення трьох змінних з подальшим наданням їм значень
let a, b, c;
a = 1;
b = 2;
c = 3;
```

JavaScript - мова з динамічною типізацією, тому оголошеній змінній в іншому місці програми можна надати значення іншого типу.

Приклад:

```
var message;
message = "Привіт!";
message = 4;
```

Одну змінну не можна кілька разів оголосити через let. Такий код призведе до помилки (Рис. 4.1):

```
let a = 1;
alert(a);
let a = 2;
alert(a);
```

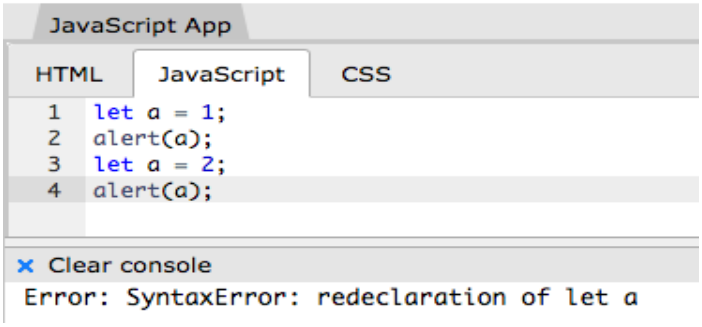


Рис. 4.1. Приклад виконання сценарію.

Можна або ввести дві різні змінні:

```
let a = 1;
alert(a);
let b = 2;
alert(b);
```

Або оголосити змінну a, а потім виконувати операції з нею:

```
let a;
a = 1;
alert(a);
a = 2;
alert(a);
```

Назва величини:

- має містити лише літери латиниці, цифри і символи "_" та "$", починатися з літери латиниці або символів "_" чи "$",
- не може збігатися з жодним з таких зарезервованих слів:
  break case class catch const continue debugger default delete do else export extends false finally for function if import in instance of let new nul return super switch this throw true try typeof var void while with yield

Наприклад, `test, _test, $var _my_test1`.

Якщо ім'я містить кілька слів, зазвичай використовується *верблюжа нотація (CamelCase),* тобто слова слідують одне за одним, де кожне наступне слово починається з великої літери:

```
myVeryLongName (lowerCamelCase)
MyVeryLongName (UpperCamelCase)
```

Типи даних

1. Число (number)

- *ціле число* допускає використання різних форматів запису значення:
  25 - запис цілого числа у десятковій системі числення;
  0137 - запис цілого числа у вісімковій системі числення;
  0xFF - запис цілого числа у шістнадцятковій системі числення;
- *дійсне число* допускає використання різних форматів запису значення:
  386.7 - запис дійсного числа с рухомою десятковою крапкою;
  25e5 або 25E5 - запис дійсного числа 25 · 105 в науковій нотації;

Спеціальні числові значення: Infinity, -Infinity, NaN.

Infinity - математична нескінченність ∞. Це спеціальне значення, що є більшим за будь-яке число.

Може бути отримане як результат ділення на нуль:

```
alert(1 / 0); // Infinity
```

Можна безпосередньо посилатися на нього:

```
alert(Infinity); // Infinity
```

NaN (Not a Number) - помилка обчислення, результат неправильної або невизначеної математичної операції, наприклад:

```
alert("not a number" / 2); // NaN, таке ділення є помилковим
```

Будь-яка подальша математична операція з NaN повертає NaN:

```
alert( NaN + 1 ); // NaN
alert( 3 * NaN ); // NaN
alert( "not a number" / 2 - 1 ); // NaN
```

Є лише один виняток: результатом операції NaN ** 0 буде 1.

Математичні операції в JavaScript є “безпечними”. Можна ділити на нуль, звертатися до нечислового рядка як до числа тощо.

Виконання скрипту ніколи не зупиниться з фатальною помилкою. У найгіршому випадку в результаті буде отримано NaN.

Спеціальні числові значення формально належать до типу “number”, хоча вони не є числами у загальноприйнятому розумінні.

2. BigInt

У JavaScript, тип “number” не може містити числа більші за $(2^{53}-1)$ (це 9007199254740991), або менші за $-(2^{53}-1)$ для від’ємних чисел. Це технічне обмеження, спричинене їхньою внутрішньою реалізацією.

Для більшості потреб цього достатньо, але бувають випадки, коли потрібні дійсно великі числа, наприклад, для криптографії.

Тип BigInt призначений для подання цілих чисел довільної довжини.

Значення з типом BigInt створюється через додавання n у кінець цілого числа:

```
// буква "n" у кінці означає, що це число типу BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

3. Рядок (string)

Рядок у JavaScript – це послідовність символів, взята у лапки.

Рядок може містити нуль символів (бути порожнім), один символ або більше.

let str = "Привіт";

Можна використовувати три типи лапок:

1. Подвійні лапки: `"Привіт"`
2. Одинарні лапки: `'Привіт'`
3. Зворотні лапки: `` `Привіт` ``

Подвійні та одинарні лапки є “звичайними”.

Зворотні лапки дають змогу вбудовувати змінні та вирази в рядок, беручі їх в ${…}

Приклад:

```
let name = "Іван";
// вбудована змінна
alert(`Привіт, ${name}е!`); // Привіт, Іване!
// вбудований вираз
alert(`результат: ${1 + 2}`); // результат: 3
alert("результат: ${1 + 2}"); // результат: ${1 + 2} (подвійні лапки не мають ніякого впливу)
```

4. Логічний тип (boolean)

Дані цього типу набувають двох значень: *true (істина)* та *false (хиба)*. Наприклад:

```
let nameFieldChecked = true; // так, ім'я було перевірене
let ageFieldChecked = false; // ні, вік не був перевіреним
```

Логічне значення також можна отримати як результат порівняння:

```
let isGreater = 4 > 1;
alert(isGreater); // true (результат порівняння - "так")
```

5. Значення null

Спеціальне значення null не належить до жодного з типів, воно формує окремий власний тип, який містить лише значення null:

```
let age = null;
```

В JavaScript null не є "посиланням на об'єкт, якого не існує" або "покажчиком на null", як може бути в інших мовах програмування.

Це спеціальне значення, яке подає "нічого", "порожнє" або "невідоме значення".

У наведеному прикладі зазначено, що значення змінної age невідоме.

6. Значення undefined

Спеціальне значення undefined також подає власний тип, подібний до "null".

undefined означає, що "значення не присвоєно".

Якщо змінна оголошена, але їй не присвоєне якесь значення, тоді значення такої змінної буде undefined:

```
let age;
alert(age); // покаже "undefined"
```

Як правило, null використовується для присвоєння змінній значення "порожнє" або "невідоме", а undefined зарезервоване для позначення початкового значення для ненаданих значень.

7. Об'єкти (object)

Тип object є особливим типом.

Усі інші типи називаються "примітивами", тому що їхні значення можуть містити тільки один елемент (це може бути рядок, число, або будь-що інше).

В об'єктах зберігаються колекції даних і більш складні структури.

8. Символи (symbol)

Тип symbol використовується для створення унікальних ідентифікаторів в об'єктах.

Організація взаємодії з користувачем. функції alert, prompt, confirm

Alert

Синтаксис:

alert (text);

функція виводить на екран модальне вікно з текстовим повідомленням та чекає, доки користувач не натисне кнопку “ОК” (Рис. 4.2).

Приклад:

```
alert("Привіт");
```

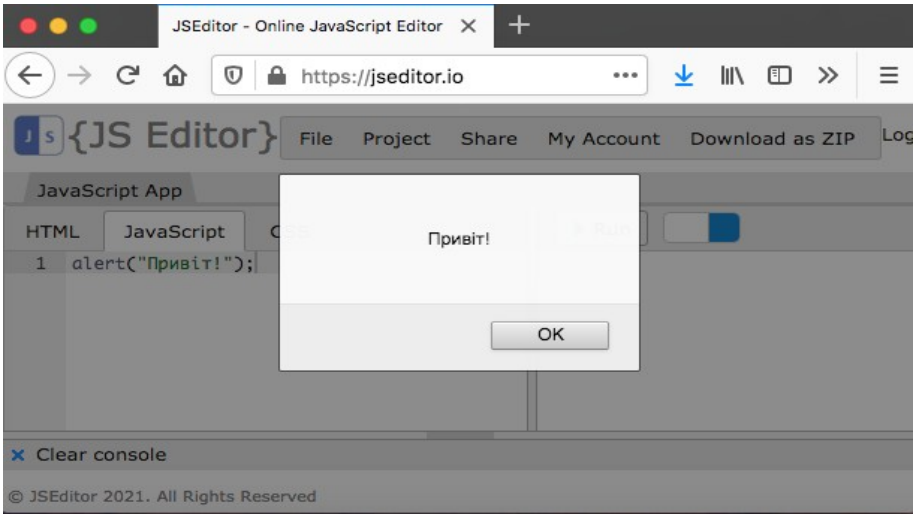


Рис. 4.2. Приклад виконання сценарію.

prompt

Синтаксис:

result = prompt(text, [default]);

Функція виводить на екран модальне вікно з текстовим повідомленням, полем, куди користувач може ввести текст, та кнопками ОК/Скасувати.

text - текстове повідомлення, яке буде відображатися для користувача.

default - необов’язковий параметр - початкове значення для поля введення тексту.

Квадратні дужки в синтаксисі [...] означають, що цей параметр є необов’язковим.

Користувач може щось ввести у поле введення і натиснути ОК. Тоді result набуде значення введеного тексту. Якщо користувач натисне “Скасувати” або клавішу Esc, result набуде значення null.

Виклик prompt повертає текст із поля введення або null, якщо введення було скасовано.

Приклад:

```
let age = prompt('Скільки вам років?', 100);
alert(`Вам ${age} років!`); // Вам 100 років!
```

(Рис. 4.3)

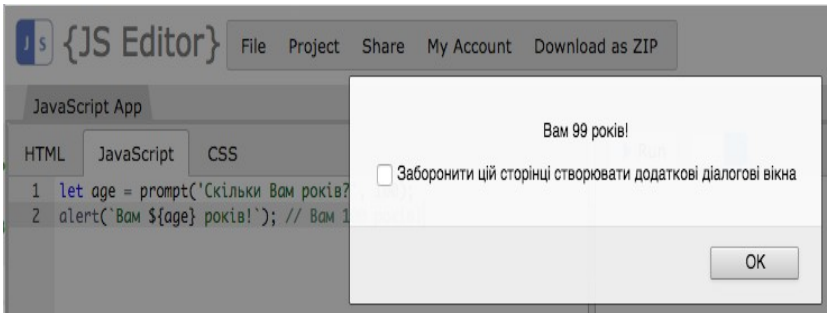


Рис. 4.3. Приклад виконання сценарію.

confirm

Синтаксис:

result = confirm(question);

Функція виводить на екран модальне вікно з питанням question та двома кнопками: ОК та Скасувати.

Результат: true, якщо натиснути кнопку ОК, інакше - false.

Приклад:

```
let isStud = confirm("Ви студент?");
alert( isStud ); // true, якщо натиснута OK
```
(Рис. 4.4)

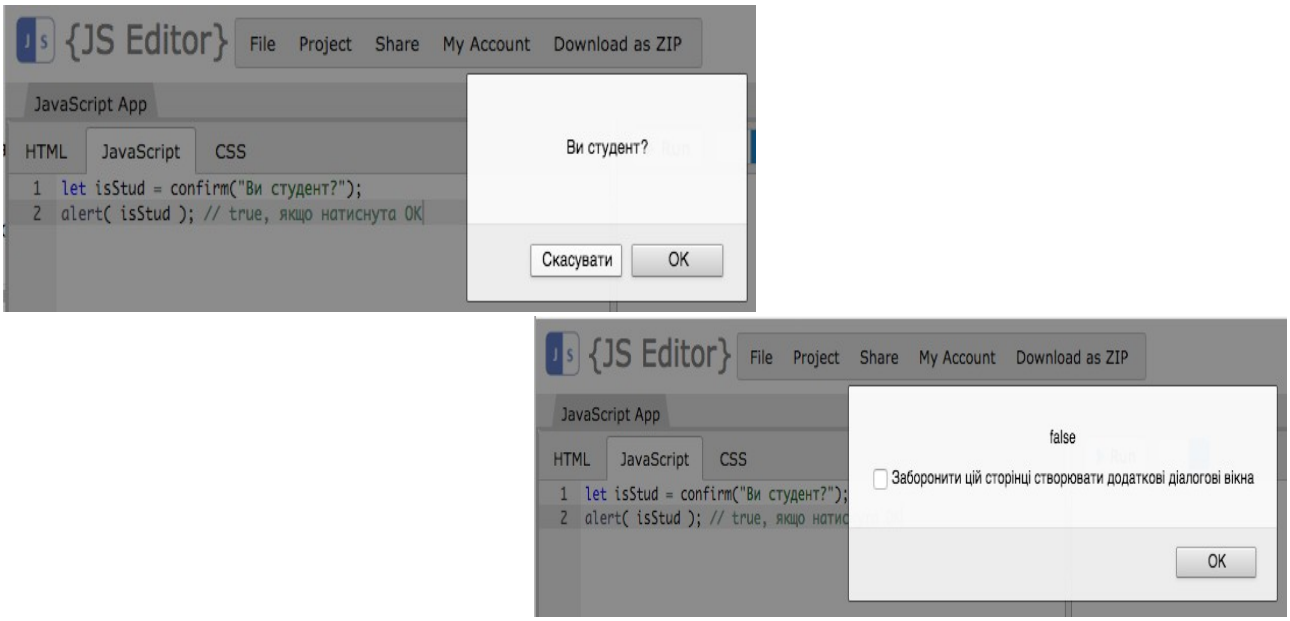


Рис. 4.4. Приклад виконання сценарію.

Обмеження розглянутих функцій:

- Точне розташування модального вікна визначається браузером. Зазвичай це в центрі.
- Точний вигляд вікна залежить від браузера.

Приклад.

Вебсторінка, яка запитує ім’я та виводить його.

JavaScript-код (Рис. 4.5):

```
let name = prompt("Як Вас звати?", "");
alert(name);
```

```
Вся сторінка:
<!DOCTYPE html>
<html>
<body>
  <script>
    'use strict';
    let name = prompt("Як Вас звати?", "");
    alert(name);
  </script>
</body>
</html>
```

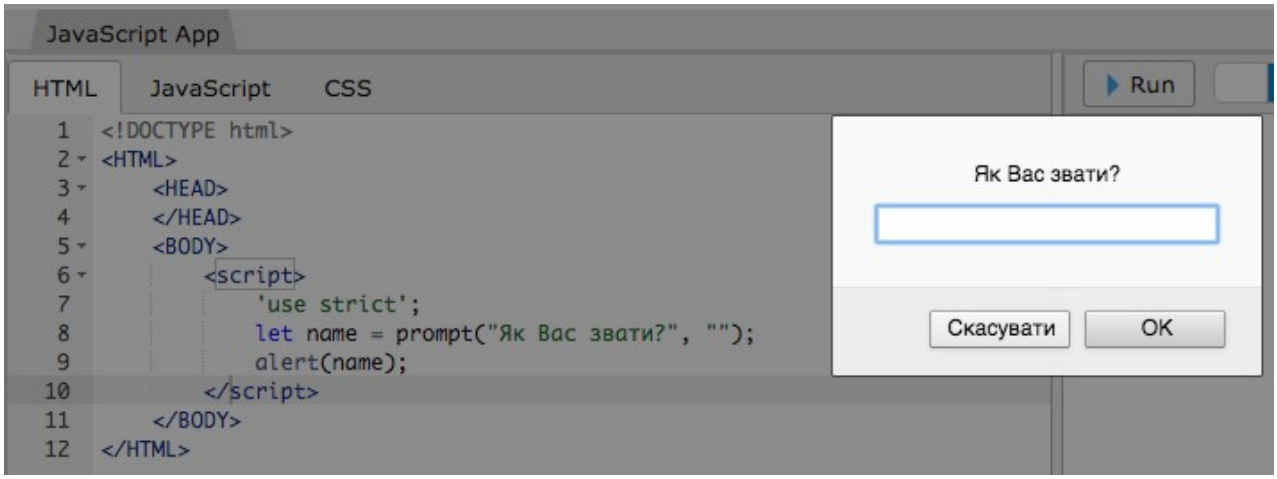


Рис. 4.5. Приклад виконання сценарію.

Перетворення типу

Переважно оператори та функції автоматично перетворюють операнди або аргументи, які їм надаються, на потрібний тип.

Наприклад, alert автоматично перетворює будь-яке значення в рядок, щоб показати його. Математичні операції перетворюють значення на числа.

Є випадки, коли потрібно явно перетворити значення на певний тип.

Три найпоширеніші перетворення типів - це перетворення на число, на рядок та на логічний тип.

Перетворення на рядок – відбувається при виведенні даних. Може бути виконане за допомогою функції **String(value)**.

Перетворення на число – відбувається в математичних операціях. Може бути виконане за допомогою функції **Number(value).**

Таблиця 4.1

Правила перетворення на числа

Значення	**Результат**
undefined	NaN
null	0
true та false	0 та 1
string	Пропуски на початку та з кінця видаляються. Якщо рядок, що залишився в результаті, порожній, то результатом є 0. В іншому випадку число “читається” з рядка. Помилка дає NaN.

Перетворення на логічний тип - відбувається в логічних операціях. Може бути виконане явно за допомогою функції **Boolean(value).**

Таблиця 4.2.

Правила перетворення на логічні значення:

Значення	**Результат**
0, null, undefined, NaN, ""	false
будь-які інші значення	true

Вирази утворюються з літералів, назв змінних, знаків операцій і дужок.

В результаті обчислення виразу отримують значення, яке може бути числом, рядком або логічним значенням.

Наприклад, у виразі a*b: a і b називають операндами, * - знаком операції або оператором.

Оператори

Унарні (для одного аргументу) оператори застосовують для зміни знаку, заперечення, збільшення (інкременту) чи зменшення (декременту):

- зміна знаку на протилежний;
! заперечення (логічної змінної);
++ збільшення на 1 цілого значення змінної;
-- зменшення на 1 цілого значення змінної.

Перші два оператори записують як префікс (перед операндом), останні два - і як префікс, і як суфікс (після операнда).

Бінарні оператори записують між двома операндами. У мові JavaScript передбачено такі бінарні оператори:

- віднімання;
+ додавання;
* множення;
/ ділення;
% залишок від ділення;
** піднесення до степеня.

Оператори для роботи з окремими бітами:

& і;
| або;
^ або ... або ...;
~ заперечення.

Оператори зсуву:

>> зсув праворуч;
<< зсув ліворуч;
>>> зсув праворуч з заповненням звільнених розрядів нулями.

Оператори зсуву і для роботи з окремими бітами перед визначенням результату перетворюють величини змінних у 32-розрядні (у двійковій системі) цілі числа.

Особливості операторів JavaScript

Об'єднання рядків через бінарний плюс

Бінарний плюс, застосований до рядків, об'єднує їх:

Приклад:

```
let s = 'мій_' + 'рядок';
alert(s); // мій_рядок
```

Якщо один з операндів є рядком, інший також перетворюється на рядок:

Приклад:

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```

Приклад:

```
alert(2 + 2 + '1' ); // "41", а не "221"
```

Тут оператори виконуються один за одним. Перший + додає два числа і повертає 4; наступний + вже додає (об'єднує) попередній результат із рядком 1.

Приклад:

```
alert('1' + 2 + 2); // "122", а не "14"
```

Перший операнд – рядок, тому компілятор також опрацьовує інші два операнди як рядки. Операнд 2 приєднується (конкатенується) до '1', тому в результаті буде '1' + 2 = "12", а потім - "12" + 2 = "122".

Так працює з рядками лише бінарний плюс. Інші арифметичні оператори працюють тільки з числами й завжди перетворюють свої операнди на числа.

Приклад:

```
alert( 6 - '2' ); // 4, '2' перетворюється на число
alert( '6' / '2' ); // 3, обидва операнди перетворюються на числа
```

Числове перетворення через унарний плюс

Унарний плюс, або оператор плюс +, застосований до одного операнда, не впливає на операнд, який є числом.

Якщо операнд не є числом, унарний плюс перетворює його на число.

Приклад:

```
// Немає впливу на числа
let x = 1;
alert( +x ); // 1
let y = -2;
alert( +y ); // -2
// Перетворення нечислових значень
alert( +true ); // 1
```

```
alert( +"" );   // 0
```
Унарний плюс працює як і Number(...), але має коротший вигляд.

Приклади застосування унарного та бінарного плюсів.
Додавання як рядків:
```
let value1 = "2";
let value2 = "3";
alert( value1 + value2 ); // "23", бінарний плюс об'єднує рядки

Додавання як чисел:
let value1 = "2";
let value2 = "3";
// обидва значення перетворюються на числа перед застосуванням бінарного плюса
alert( +value1 + +value2 ); // 5
// довший варіант
// alert( Number(value1) + Number(value2) ); // 5
```

Пріоритет унарного плюса вищий, ніж бінарного, тому спочатку до значень застосовуються унарні плюси, перетворюючи їх на числа, після чого числа додаються з використанням бінарного плюса.

Матеріали для самоосвіти
Повна таблиця пріоритетів операцій у порядку спадання пріоритету
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence?retiredLocale=uk

Таблиця 4.3.

Таблиця пріоритетів операцій у порядку спадання пріоритету

Назва	**Позначення**
інкремент	++
декремент	--
заперечення	!
унарний плюс	+
унарний мінус	-
піднесення до степеня	**
множення	*
ділення	/
залишок від ділення	%
додавання	+
віднімання	-
порівняння	<, >, <=, >=
рівність	==
нерівність	!=
логічне «і»	&&
логічне «або»	||
присвоєння значення	=, +=, -=, *=, /=, %=, !=

Присвоєння

Присвоєння = оператор з низьким пріоритетом.

Тому при присвоєнні значення змінній, наприклад, x = 2 * 2 + 1, спочатку виконуються обчислення, а потім виконується присвоєння = зі збереженням результату в x.

```
let x = 2 * 2 + 1;
alert( x ); // 5
```

Присвоєння = повертає результат

x = значення записує значення у x, а потім повертає його.

Приклад. Використання присвоєння як частини складнішого виразу:

```
let a = 1;
let b = 2;
let c = 3 - (a = b + 1);
alert( a ); // 3
alert( c ); // 0
```

Результат виразу (a = b + 1) є значенням, яке присвоювалося змінній a (тобто 3). Потім воно використовується для подальших обчислень.

Ланцюгові присвоєння

Приклад:

```
let a, b, c;
a = b = c = 2 + 2;
alert( a ); // 4
alert( b ); // 4
alert( c ); // 4
```

Ланцюгове присвоєння виконується справа наліво. Спочатку обчислюється найправіший вираз 2 + 2, а потім результат присвоюється змінним ліворуч: c, b та a. Зрештою всі змінні мають спільне значення.

Для покращення читабельності коду краще розділяти подібні конструкції на декілька рядків:

```
c = 2 + 2;
b = c;
a = c;
```

Так легше прочитати, особливо коли швидко переглядати код.

Оператор "модифікувати та надати значення"

Використовується, коли потрібно застосувати оператор до змінної і зберегти новий результат у ту ж саму змінну.

Приклад:

```
let n = 2;
n = n + 5;
n = n * 2;
```

Цей запис можна скоротити за допомогою операторів += та *=:

```
let n = 2;
n += 5; // n = 7 (n = n + 5)
n *= 2; // n = 14 (n = n * 2)
alert( n ); // 14
```

Короткі оператори "модифікувати та надати значення для арифметичних та побітових операторів:

= звичайне присвоєння;
+= збільшення числової величини або злиття рядків;
-= зменшення числової величини;
*= множення;
/= ділення;
%= залишок від ділення;
>>= зсув праворуч;
>>>= зсув праворуч з заповненням звільнених розрядів нулями;
<<= зсув ліворуч;
|= «або»;
&= «і»;
^= «або ... або ...».

Ці оператори мають такий же пріоритет, як і звичайне присвоєння, тому вони виконуються після більшості інших обчислень:

```
let n = 2; n *= 3 + 5;
alert( n ); // 16  (права частина обчислюється першою, так само, як n *= 8)
```

Інкремент/декремент

Збільшення або зменшення на одиницю є однією з найпоширеніших числових операцій, тому для цього є спеціальні оператори:

Інкремент ++ збільшує змінну на 1:

```
let counter = 2;
counter++;     // працює, як counter = counter + 1, але запис коротше
alert( counter ); // 3
```

Декремент **--** зменшує змінну на 1:

```
let counter = 2;
counter--;     // працює, як counter = counter - 1, але запис коротше
alert( counter ); // 1
```

Інкремент/декремент можуть застосовуватися лише до змінних.

Використання їх із значенням (5++) призведе до помилки.

Оператори ++ та -- можуть розташовуватися до або після змінної.

- *Постфіксна форма* – оператор вказується після змінної: counter++.
- *Префіксна форма* – оператор вказується перед змінною: ++counter.

Обидві інструкції збільшують counter на 1.

Різниця – у значеннях, які повертають оператори ++/--.

Префіксна форма інкременту/декременту повертає нове значення, постфіксна форма інкременту/декременту повертає старе значення (до збільшення / зменшення).

Приклад:

```
let counter = 1;
let a = ++counter;
alert(a); // 2
```

Префіксна форма ++counter збільшує counter та повертає нове значення, 2. Отже, alert показує 2.

```
let counter = 1;
let a = counter++; // змінили ++counter на counter++
alert(a); // 1
```

Постфіксна форма counter++ також збільшує counter, але повертає старе значення (до інкременту). Отже, alert показує 1.

Якщо результат збільшення/зменшення не використовується, немає різниці, яку форму використовувати:

```
let counter = 0;
```

```
counter++;
++counter;
alert( counter ); // 2, у рядках вище робиться одне і те ж саме
```

Якщо потрібно збільшити значення та негайно використати результат оператора, використовується префіксна форма:

```
let counter = 0;
alert( ++counter ); // 1
```

Якщо потрібно збільшити значення, але використати його попереднє значення, використовується постфіксна форма:

```
let counter = 0;
alert( counter++ ); // 0
```

Оператори ++/-- також можуть використовуватися всередині виразів. Їхній пріоритет вищий за більшість інших арифметичних операцій.

Приклад:

Порівняємо два результати інкременту:

```
let counter = 1;
alert( 2 * ++counter ); // 4
let counter = 1;
alert( 2 * counter++ ); // 2, тому що counter++ повертає "старе" значення
```

Такий запис допустимий, але робить код менш читабельним.

Рекомендується використовувати стиль стиль “одна лінія – одна дія”:

```
let counter = 1;
alert( 2 * counter );
counter++;
```

Оператори порівняння використовують для порівняння величин змінних. Результат - true або false. Їх використовують переважно в умовних операторах.

Перелік операторів порівняння з указанням умови повертання true:

>	лівий операнд більший за правий;
>=	лівий операнд не менший за правий;
<	лівий операнд менший за правий;
<=	лівий операнд не більший за правий;
==	величини операндів збігаються;
!=	величини операндів різні.

Логічні оператори

| | оператор «або», який повертає true, якщо хоча б один з операндів дорівнює true;

&& оператор «і», який повертає true, якщо обидва операнди дорівнюють true.

Порівняння рядків

У JavaScript рядки порівнюються посимвольно, тобто використовується так званий “алфавітний” або “лексикографічний” порядок. Використовується кодування Unicode.

Приклад:

```
alert( 'Я' > 'А' ); // true
alert( 'Соки' > 'Сода' ); // true
alert( 'Комар' > 'Кома' ); // true
```

Порівняння різних типів

При порівнянні значень різних типів JavaScript конвертує ці значення в числа.

Приклад:

```
alert( '2' > 1 ); // true, рядок '2' стає числом 2
alert( '01' == 1 ); // true, рядок '01' стає числом 1
```

Логічне значення true стає 1, а false - 0.

Приклад:

```
alert( true == 1 ); // true
alert( false == 0 ); // true
```

Можливі такі ситуації: два значення рівні; одне з них true як логічне значення, а інше - false.

Приклад:

```
let a = 0;
alert( Boolean(a) ); // false
let b = "0";
alert( Boolean(b) ); // true
alert(a == b); // true!
```

З погляду JavaScript, результат очікуваний. Порівняння перетворює значення на числа (тому "0" стає 0), тоді як явне перетворення за допомогою Boolean використовує інший набір правил.

Строге порівняння

Звичайний оператор порівняння == перевіряє значення на рівність з перетворенням типів.

Тому, наприклад, він не відрізняє 0 або порожній рядок від false:

```
alert( 0 == false ); // true
alert( '' == false ); // true
```

Як відрізнити 0 від false?

Оператор строгої рівності **===** перевіряє рівність без перетворення типів.

Якщо a і b мають різні типи, то перевірка a === b негайно поверне результат false без спроби їхнього перетворення.

Приклад:

```
alert( 0 === false ); // false, тому що порівнюються різні типи
```

Існує також оператор строгої нерівності !==, аналогічний до !=.

Оператор строгої рівності робить код більш зрозумілим і залишає менше місця для помилок.

Порівняння з null та undefined

```
alert( null === undefined ); // false
alert( null == undefined ); // true
```

При строгому порівнянні === ці значення різні, тому що різні їхні типи.

При нестрогому порівнянні == ці значення рівні. Водночас ці значення не рівні значенням інших типів. Це спеціальне правило мови.

Під час використання математичних операторів та інших операторів порівняння < > <= >= значення null/undefined конвертуються в числа: null стає 0, а undefined стає NaN.

Логічні оператори

В JavaScript існує чотири логічні оператори:

|| АБО,
&& І,
! НЕ,
?? оператор null-об'єднання.

Ці оператори можуть бути застосовані до значень будь-якого типу, не тільки булевих. Їх результати також можуть бути будь-якого типу.

Матеріали для самоосвіти:
https://uk.javascript.info/logical-operators
https://uk.javascript.info/nullish-coalescing-operator

|| (АБО)

```
result = a || b;
```

Особливості JavaScript.

Задано кілька значень, розділених оператором АБО:

```
result = value1 || value2 || value3;
```

Виконання оператора || :

- операнди обчислюються зліва направо;
- значення кожного операнда на перетворюється на булеве; якщо результат true, виконання припиняється і повертається початкове значення цього операнда;
- якщо всі операнди виявились false, повертається останній операнд.
- значення повертається у первісному вигляді без конвертації.

Отже, ланцюжок з || повертає перше істинне значення або останнє, якщо істинного значення не знайдено.

Приклад:
```
alert( 1 || 0 ); // 1 (1 є істинним)
alert( null || 1 ); // 1 (1 є першим істинним значенням)
alert( null || 0 || 1 ); // 1 (перше істинне значення)
alert( undefined || null || 0 ); // 0 (усі хибні, повертається останнє значення)
```

Отримання першого істинного значення зі списку змінних або виразів.

Приклад. Є змінні firstName, lastName, nickName.
Усі необов'язкові (тобто можуть бути невизначеними або мати хибні значення).
Використаємо ||, щоб вибрати ту змінну, яка має дані, і виведемо її (або рядок "Анонім", якщо жодна змінна не має даних):

```
let firstName = "";
let lastName = "";
let nickName = "Хтось";
alert( firstName || lastName || nickName || "Анонім"); // Хтось
```

Якщо б усі змінні мали порожні рядки, вивелось би слово "Анонім".

&& (I)

```
result = a && b;
```

І "&&" шукає перше хибне значення

Задано декілька значень, об'єднаних кількома І:

```
result = value1 && value2 && value3;
```

Виконання оператора &&:

- операнди обчислюються зліва направо;
- кожен операнд перетворюється на булевий. Якщо результат false, виконання припиняється і повертається початкове значення цього операнда;
- якщо всі операнди були були істинні, повертається останній операнд.

Отже, && повертає перше хибне значення, або останнє значення, якщо жодного хибного не було знайдено.

Таким чином, І (&&) повертає перше хибне значення,
АБО (||) повертає перше істинне значення.

Приклади:
якщо перший операнд істинний, І повертає другий операнд:

```
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5
```

якщо перший операнд хибний, І повертає саме його. Другий операнд ігнорується

```
alert( null && 5 ); // null
alert( 0 && "неважливо" ); // 0
```

повертається перше хибне значення:

```
alert( 1 && 2 && null && 3 ); // null
```

Коли всі значення є істинними, повертається останнє значення:

```
alert( 1 && 2 && 3 ); // 3, останнє
```

! (НЕ)

```
result = !value;
```

Виконання оператора:

- операнд перетворюється на булевий тип: true/false;
- повертається зворотне значення.

Приклад:

```
alert( !true ); // false
alert( !0 ); // true
```

Подвійний !! іноді використовується для перетворення значення на булевий тип:

```
alert( !!"не порожній рядок" ); // true
alert( !!null ); // false
```

Тобто, перший НЕ перетворює значення на булеве і повертає зворотне, а другий НЕ інвертує його знову. Отже, відбувається перетворення значень на булевий тип.

Інший спосіб такого перетворення – вбудована функція Boolean:

```
alert( Boolean("не порожній рядок") ); // true
alert( Boolean(null) ); // false
```

Пріоритет ! є найвищим серед усіх логічних операторів, тому він завжди виконується першим, перед && або ||.

Умовний оператор:

if (умова) оператор1

(скорочена форма)

if (умова) оператор1 else оператор2

(повна форма)

Виконання оператора.

При справдженні умови виконується оператор1.

Для повної форми при хибності умови виконується оператор2

Якщо оператори містять більше однієї інструкції, їх необхідно брати в фігурні дужки.

Для покращення читабельності коду фігурні дужки рекомендується використовувати завжди.

Приклад.

```
let year = prompt('В якому році була створена мова JavaScript?', '');
if (year == 1995) alert('Правильно!' )

if (year == 1995) {
  alert("Правильно!" );
  alert( "Ви молодець!" );
}
```

Приклад.
```
let year = prompt('В якому році була створена мова JavaScript?',
'');
if (year == 1995) {
  alert('Правильно!' );
} else {
  alert('Ви помилились...' );
}
```

Блок else if в умовному операторі дає змогу перевірити кілька варіантів умови.
Приклад:
```
let year = prompt('В якому році була створена мова JavaScript?',
'');
if (year < 1995) {
  alert( 'Зарано...' );
} else if (year > 1995) {
  alert( 'Запізно' );
} else {
  alert( 'Саме так!' );
}
```
Може бути більше else if блоків. Останній блок else є необов'язковим.

Умовний оператор "?"

Використовується для надання значення змінній в залежності від умови.
Синтаксис:
let result = умова ? значення1 : значення2;
Виконання оператора:

обчислюється умова: якщо вона є істинною, повертається значення1, інакше – значення2.

Приклад.
```
let accessAllowed = (age > 18) ? true : false;
```
можна записати простіше:
```
let accessAllowed = age > 18;
```
Послідовність операторів знака питання ? може повернути значення, яке залежить від більш ніж однієї умови.
Переписати if, використовуючи '?':
Приклад 1.
```
let result;
if (a + b < 4) {
  result = 'Нижче';
} else {
  result = 'Вище';
}
let result = (a + b < 4) ? 'Нижче' : 'Вище';
```
Приклад 2.
```
let message;
if (login == 'Працівник') {
```

```
  message = 'Привіт';
} else if (login == 'Директор') {
  message = 'Вітаю';
} else if (login == '') {
  message = 'Немає логіну';
} else {
  message = '';
}
let message = (login == 'Працівник') ? 'Привіт' :
(login == 'Директор') ? 'Вітаю' :
(login == '') ? 'Немає логіну' :
 '';
```

Оператор вибору:

switch (вираз) {
case значення: оператори [break];
case значення: оператори [break];
...
default: оператори [break]}

Виконання оператора.

1. Значення виразу перевіряється на строгу рівність (===) значенню із першого блоку case, потім значенню із другого блоку і так далі.
2. Якщо строго рівне значення знайдено, то виконується код із відповідного блоку case до найближчого break або до кінця всієї конструкції switch.
3. Якщо жодне case-значення не збігається – виконується код із блоку default (якщо він присутній).

Приклад.

```
let a = 2 + 2;
switch (a) {
  case 3:
    alert( 'Замало' );
    break;
  case 4:
    alert( ‘Правильно!' );
    break;
  case 5:
    alert( 'Забагато' );
    break;
  default:
    alert('Не знаю таких значень' );
}
```

Оператори циклу

використовують для багаторазового виконання послідовності операторів, поки певний логічний вираз не стане хибним.

Схожий на оператори циклу оператор ітерації **for…in**, який використовують при роботі з об'єктами.

JavaScript має 3 форми оператора циклу:

- for (ініціалізація; логічний вираз; зміна) оператор
- while (логічний вираз) оператор
- do оператор while (логічний вираз)

for (ініціалізація; лог_вираз; зміна) оператор

Виконання оператора.

1. Здійснюється ініціалізація: лічильнику (лічильникам) циклу надається початкове значення.
2. Встановлюється істинність логічного виразу. При хибності відбувається перехід до наступного за for оператора.
3. Виконується оператор.
4. Виконується зміна (зазвичай це збільшення або зменшення лічильника) і перехід до пункту 2.

Приклад.

```
for (let i = 0; i < 3; i++) { // вивед. 0,  1,  2
  alert(i);
}
```

Всередині циклу оголошена змінна і, яка виконує функцію лічильника. Це так зване «вбудоване» оголошення змінної. Такі змінні доступні лише всередині циклу.

```
alert(i); // помилка, немає такої змінної
```

Замість оголошення нової змінної можна використовувати наявну:

```
let i = 0;
for (i = 0; i < 3; i++) {
  alert(i); // 0, 1, 2
}
alert(i); // 3, змінна доступна, бо оголошена поза циклом
```

Будь-яку частину for можна пропустити (дослідити самостійно)

while (логічний вираз) оператор

Виконання оператора.

1. Встановлюється істинність логічного виразу. При хибності відбувається перехід до наступного за while оператора.
2. Виконується оператор і перехід до пункту 1.

Якщо оператор (тіло циклу) складається більш ніж з одної інструкції, його треба брати в фігурні дужки.

Одне виконання циклу називається *ітерацією*.

Приклад.
```
let i = 0;
while (i < 3) { // показується 0, далі 1, потім 2
  alert( i );
  i++;
}
```

Умовою циклу може бути будь-який вираз або змінна, а не тільки порівняння (a < 5 чи b !== 10). Умова виконується і конвертується у логічне значення.

Приклад. Коротший спосіб написання while (i != 0) відповідає while (i):
```
let i = 3;
while (i) { // коли i = 0, умова стане хибною, і цикл зупиниться
  alert( i );
  i--;
}
```

do оператор while (логічний вираз)

Виконання оператора.

1. Виконується *оператор - тіло циклу.*
2. Встановлюється істинність логічного виразу. При хибності відбувається перехід до наступного за do…while оператора.
3. Перехід до пункту 1.

На відміну від попереднього оператора циклу, цей виконує оператор хоча б один раз.

Приклад.
```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3)
```
Приклад. Вивести парні числа від 2 до 10, використовуючи цикл for.
```
for (let i = 2; i <= 10; i++) {
  if (i % 2 == 0) {
    alert( i );
  }
}
```
Приклад. Замінити цикл for на while так, щоб виведення не змінилось.
```
for (let i = 0; i < 3; i++) {
  alert( `число ${i}!` );
}
let i = 0;
while (i < 3) {
  alert( `число ${i}!` );
  i++;
}
```

Функції

У багатьох випадках функції є способом зв'язати разом кілька вказівок для того, щоб викликати їх з різних рядків коду.

В JavaScript є вбудовані функції, крім цих функцій, користувач може створювати власні.

Синтаксис опису функції:

function назва ([параметр 1] [,параметр 2] [...,параметр N])
{ . . .
рядки тіла функції
. . .
[return величина]
}

Матеріали для самоосвіти:
https://uk.javascript.info/function-basics

За допомогою оператора return функція може повернути величину. Цей оператор має дві форми:

1. **return вираз**
2. **return**

return вираз завершує виконання функції і повертає величину виразу. Функцію з таким оператором викликають як *частину виразу оператора надання величини*.

return завершує виконання функції і повертає величину undefined. Функцію з таким оператором викликають як *оператор.*

Якщо тіло функції не містить оператора return, то її виконання завершується з виконанням останнього оператора тіла функції і повертається величина undefined.

Значення, які передаються в функцію в якості параметрів, копіюються в локальні змінні.

Функції мають доступ до зовнішніх змінних. Але це працює тільки зсередини назовні. Код поза функцією не має доступу до локальних змінних функції.

Функція може повертати значення. Якщо цього не відбувається, результат буде undefined.

Для того, щоб зробити код зрозумілим, рекомендується використовувати локальні змінні і параметри функції, не користуватися зовнішніми змінними.

Завжди легше зрозуміти функцію, яка отримує параметри, працює з ними і повертає результат.

Ім’я функції повинне бути коротким і чітко відображати, що робить функція.

Приклад. Функція, що повертає суму двох чисел.

```
function sum(a,b)
{   c=a+b
return c
}
```

Приклад. Функція, що повертає менше з двох чисел.

З використанням if:

```
function min(a, b) {
  if (a < b) {
    return a;
  } else {
    return b;
  }
}
```

З використанням оператора '?':

```
function min(a, b) {
  return a < b ? a : b;
}
```

5. СТРУКТУРИ ДАНИХ. МАСИВИ, РЯДКИ, ОБ'ЄКТИ

Масиви

Масив - різновид об'єкта, що призначений для зберігання пронумерованих значень і має додаткові методи для опрацювання.

Масиви переважно використовуються для зберігання впорядкованих колекцій даних, наприклад, списку товарів на сторінці, студентів у групі тощо.

Масиви налаштовані на роботу з неперервними впорядкованими даними, тому рекомендується використовувати їх саме таким чином.

Матеріали для самоосвіти:
https://uk.javascript.info/array-methods

Оголошення масиву

Для створення нового масиву вказується перелік елементів у квадратних дужках; порожній масив подається квадратними дужками без елементів.

let arr1=[];

```
let naturalSciences=['math', 'physics', 'chemistry', 'biology'];
```

Створення масиву викликом new Array():

new Array()

```
let seasons=new Array('spring','summer','fall','winter');
```

Встановити чи повернути кількість елементів масиву: властивість length

Array.length

```
alert(arr1.length); // 0
alert(naturalSciences.length);          // 4
```

Точніше, length відображає індекс останнього елемента плюс один.

```
naturalSciences.length=2;
alert(naturalSciences); // див. Рис. 5.1
```

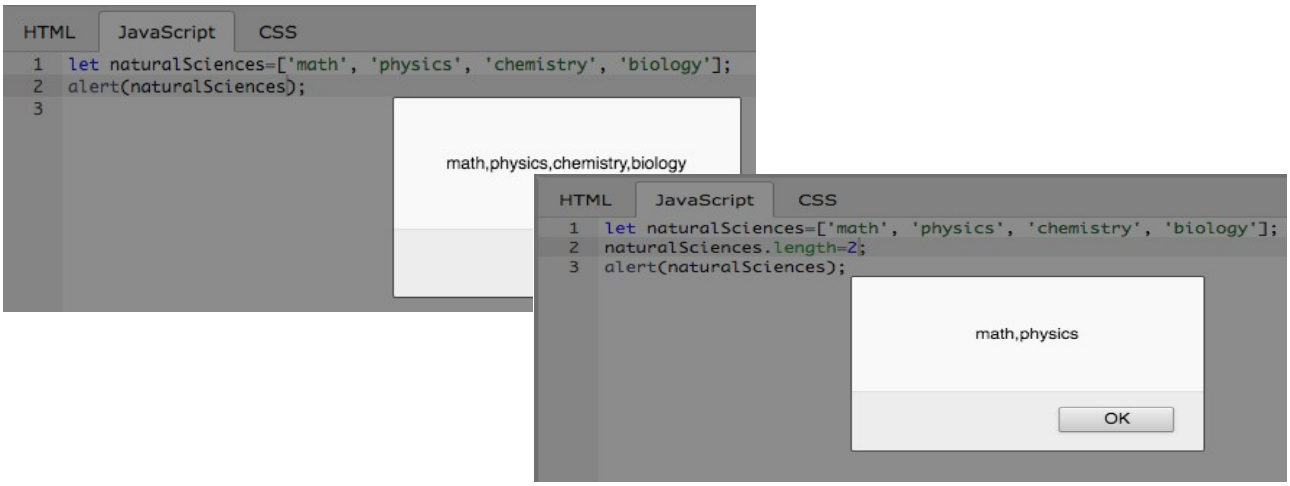


Рис. 5.1. Приклад виконання сценарію.

Елементи масиву нумеруються, починаючи з нуля.

Доступ до елемента здійснюється за його номером – ***індексом***, вказаним у квадратних дужках після імені масиву.

```
let naturalSciences=['math', 'physics', 'chemistry', 'biology'];
let arr1=[];
```

Перший елемент масиву:

```
alert(naturalSciences[0]);    // math
alert(arr1[0]);           // undefined
```

Останній елемент масиву:

```
alert(naturalSciences[naturalSciences.length-1]);      //biology
```

Зміна значення елемента масиву:

```
let naturalSciences=['math', 'physics', 'chemistry', 'biology'];
naturalSciences[0]='mathematics';
alert(naturalSciences); //  mathematics,physics,chemistry,biology
```

Додавання елемента до масиву:

```
naturalSciences[4]='Computer Science'; // Див. Рис. 5.2.
console.log(naturalSciences);
```

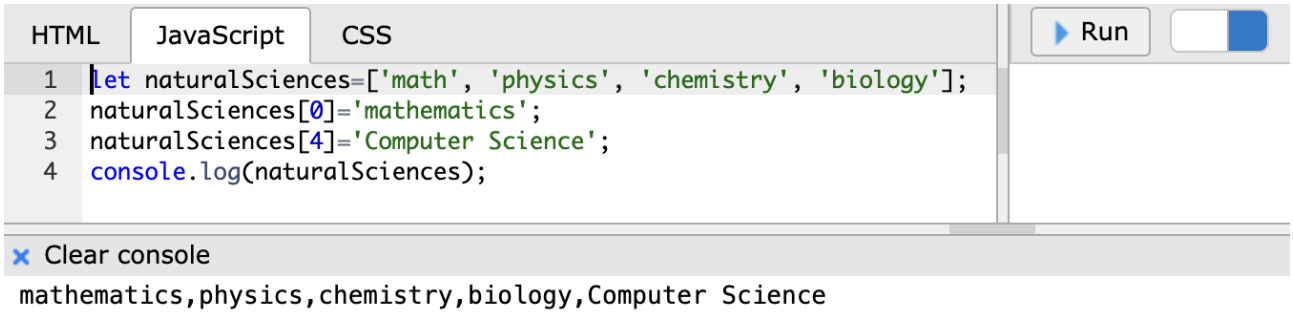


Рис. 5.2. Виведення результатів роботи програми в консоль браузера.

```
let arr1=[];
arr1[0]=10;
arr1[1]=11;
arr1[5]=15;
alert(arr1);        // 10,11,,,,15
```

Додавання елемента item в кінець масиву: push

Array.push(item)

```
arr1.push(20);
console.log(arr1);       //10,11,,,,15,20
```

Видалення останнього елемента масиву з поверненням цього елемента: pop

Array.pop()

```
console.log(arr1.pop());      //20
console.log(arr1);       // 10,11,,,,15
```

Видалення першого елемента масиву з поверненням цього елемента: shift
Array.shift()
```
let arr2=[10,20,30,40,50];
console.log(arr2);            // 10,20,30,40,50
arr2.shift();
console.log(arr2);            // 20,30,40,50
console.log(arr2.shift());    // 20
```

Додавання елемента item в початок масиву: unshift
Array.unshift(item)
```
let arr2=[10,20,30,40,50];
arr2.unshift(100);
console.log(arr2);       // 100,10,20,30,40,50
```

Пошук номера елемента item в масиві: IndexOf
Array.indexOf(item)
```
let ind=arr2.indexOf(40);
alert(ind);         //2
alert(arr2.indexOf(20)); // -1 – елемента в масиві немає
```

Видалення елемента з певним індексом: splice
Array.splice(ind, 1)
```
let arr2=[10,20,30,40,50];
arr2.splice(2,1);
  console.log(arr2);     //10,20,40,50
```

Видалення кількох (n) елементів, починаючи з певного індекса (ind): splice
Array.splice(ind, n)
```
let arr2=[10,20,30,40,50];
arr2.splice(1,3);
console.log(arr2);  //10,50
```

Створення копії масиву: slice
newArray = Array.slice();
```
let arr2=[10,20,30,40,50];
let arr22=arr2.slice()
console.log(arr22);      //10,20,30,40,50
```

slice(start,stop) повертає копію з індексами від start до stop-1
```
let arr2=[10,20,30,40,50];
let arr33=arr2.slice(0,3)
console.log(arr33);      //10,20,30
```

Упорядкування масиву: sort
Array.sort()
За замовчуванням елементи порівнюються як рядки.

```
let
naturalSciences=['mathematics','physics','chemistry','biology',
'Computer Science'];
console.log(naturalSciences.sort());
// Computer Science,biology,chemistry,mathematics,physics
```

Зміна порядку елементів масиву на зворотний: reverse

Array.reverse()

```
let
naturalSciences=['mathematics','physics','chemistry','biology',
'Computer Science'];
console.log(naturalSciences.reverse());
//Computer Science,biology,chemistry,physics,mathematics
```

Перетворення масиву в рядок: join()

Array.join([sep])

Метод повертає рядок, в якому через роздільник sep перераховані всі елементи масиву. Якщо роздільник не вказано, за замовчуванням використовується кома.

```
let arr2=[10,20,30,40,50];
console.log(arr2.join());     // 10,20,30,40,50
let str2=arr2.join("*");
console.log(str2);       // 10*20*30*40*50
console.log(str2.length);     // 14
```

Для перебору елементів масиву використовуються цикли:

Приклад.

```
let arr2=[10,20,30,40,50];
for(let i= 0; i<arr2.length; i++) {
     console.log(arr2[i]);
```

Багатовимірні масиви

Елементами масивів можуть бути інші масиви. Це можна використовувати для створення багатовимірних масивів, наприклад, матриць.

```
let matr=[[1,0,0],[0,1,0],[0,0,1]];
let trace=matr[0][0]+matr[1][1]+matr[2][2];//слід матриці
console.log('matrix trace = '+trace);
```

Знаходження сліду з використанням циклу:

```
let trace=0;
for (let i=0; i<3; i++) {
    trace+=matr[i][i];
    }
console.log('matrix trace = '+trace);
```

Приклад. Створення масиву із заданої користувачем кількості цілих випадкових чисел від 0 до 10. За замовчуванням кількість елементів масиву – 10.

```
let arr=[];
let n = prompt("Enter number of items:",10);
```

```
for (let i=0; i<n; i++) {
    arr[i]=Math.round(10*Math.random());
      }
console.log('array: '+arr);
```

Приклад. Знаходження елемента масиву, створеного як у попередньому прикладі, з випадковим індексом

```
let arr=[];
let n = prompt("Enter number of items:",10);
for (let i=0; i<n; i++) {
    arr[i]=Math.round(10*Math.random());
      }
console.log('array: '+arr);
ind=Math.floor(n*Math.random());
console.log('random index: '+ind);
console.log('item[random index]: '+arr[ind]);
```

Приклад виконання:
array: 1,3,7,0
random index: 2
item[random index]: 7

Рядки

Рядок (**String**) – це різновид об'єкту, що використовується для маніпулювання текстами.

Рядок у JavaScript – це послідовність символів, взята у лапки.

Можна використовувати три типи лапок: подвійні, одинарні, зворотні.

Способи створення рядка:
String(argument)
new String(argument)
argument - все, що може бути перетворене у рядок.
Символи в рядку нумеруються, починаючи з 0.

Найбільш поширені операції з рядками:
перевірка довжини рядка: length
str.length

```
let str='Computer Science';
console.log(str.length); // 16
```

побудова рядка за допомогою рядкової конкатенації
(+ та +=);

```
let str1='Computer';
let str2='Science';
console.log(str1+' '+str2);   // Computer Science
let str='math';
str+='ematics';
```

```
console.log(str);   //  mathematics
```

перевірка на існування або розташування підрядків: indexOf():
str.indexOf(value, [fromIndex])

Метод повертає індекс першого входження значення value в рядок str починаючи з індексу fromIndex. fromIndex - необов'язковий параметр, за замовчуванням дорівнює 0. Якщо значення не знайдено, повертається -1.

```
let str='Computer Science';
console.log(str.indexOf('en'))     // 12
console.log(str.indexOf('e'));     // 6
console.log(str.indexOf('e',13));  // 15
console.log(str.indexOf('a'));     // -1
```

отримання підрядків: substring():
str.substring(indexA[, indexB])

Метод повертає підрядок рядка, починаючи з indexA до indexB-1.

- Якщо indexA = indexB-1, повертається порожній рядок;
- якщо аргумент indexB відсутній, підрядок береться до кінця рядка;
- якщо будь-який з аргументів менше нуля, дорівнює нулю або дорівнює NaN, він буде трактуватися як рівний нулю;
- якщо будь-який з аргументів більше за довжину рядка, він буде трактуватися як рівний довжині рядка;
- якщо перший аргумент більше за другий, метод міняє їх місцями.

```
let str='Computer Science';
console.log(str.substring(9));          // Science
console.log(str.substring(9,12));  // Sci
console.log(str.substring(8,5));   // ter
```

Доступ до окремого символу рядка

Рядок можна розглядати як масивоподібний об'єкт, в якому символи мають відповідні індекси. Тоді доступ до символу рядка здійснюється за його індексом, вказаним у квадратних дужках після імені рядка.

```
let str='Computer Science';
console.log(str[0]);               // C
console.log(str[str.length-1]);    // e
```

Матеріали для самоосвіти
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String

Приклад. Сформувати масив з уведеної користувачем кількості випадкових цілих чисел від 10 до 100. Вивести масив у вікно браузера, використавши метод document.write(). Ввести ціле число з проміжку від 10 до 90. З'ясувати, чи міститься воно в масиві. Якщо так, вивести перший індекс входження. Інакше вивести відповідне повідомлення (див. Рис. 5.3).

```
<!DOCTYPE html>
<HTML>
     <HEAD>
         <script>
         let arr=[];
        let n = prompt("Введіть кількість елементів масиву:",20);
        for (let i=0; i<n; i++) {
          arr[i]=Math.round(10+90*Math.random());
            }
        document.write('Масив: '+arr);
        let value = +prompt("Введіть число від 10 до 100:",99);
        let ind=arr.indexOf(value);
        if (ind>=0) {
            alert(`Число ${value} є в масиві. Перший індекс= ${ind}`);
        }
        else {alert(`Числа ${value} в масиві немає.`);
        }
         </script>
     </HEAD>
     <BODY>
     </BODY>
</HTML>
```

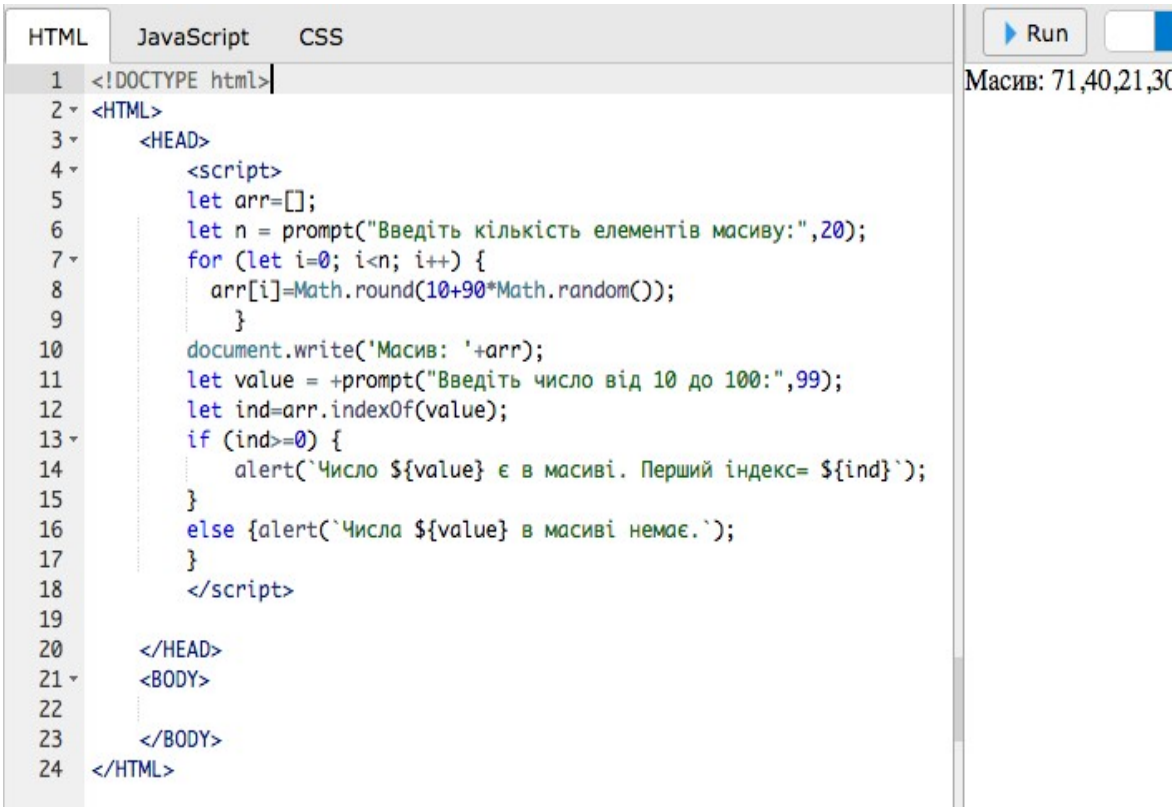


Рис. 5.3. Приклад виконання сценарію.

Приклад. Сформувати масив з уведеної користувачем кількості випадкових цілих чисел від 0 до 10. Вивести масив у вікно браузера, використавши метод document.write(). Ввести ціле число з проміжку від 0 до 10. Вивести в консоль браузера всі індекси, під якими число міститься в масиві (див. Рис. 5.4).

```
<script>
         let arr=[];
         let n = prompt("Введіть кількість елементів масиву:",20);
        for (let i=0; i<n; i++) {
          arr[i]=Math.round(10*Math.random());
            }
        document.write('Масив: '+arr);

        let value = +prompt("Введіть число від 0 до 10:",0);

          var indexes = [];
        var ind = arr.indexOf(value);
        while (ind != -1) {
        indexes.push(ind);
        ind = arr.indexOf(value, ind + 1);
        }
        console.log(`Індекси числа ${value} в масиві:`);
        console.log(indexes);
    </script>
```

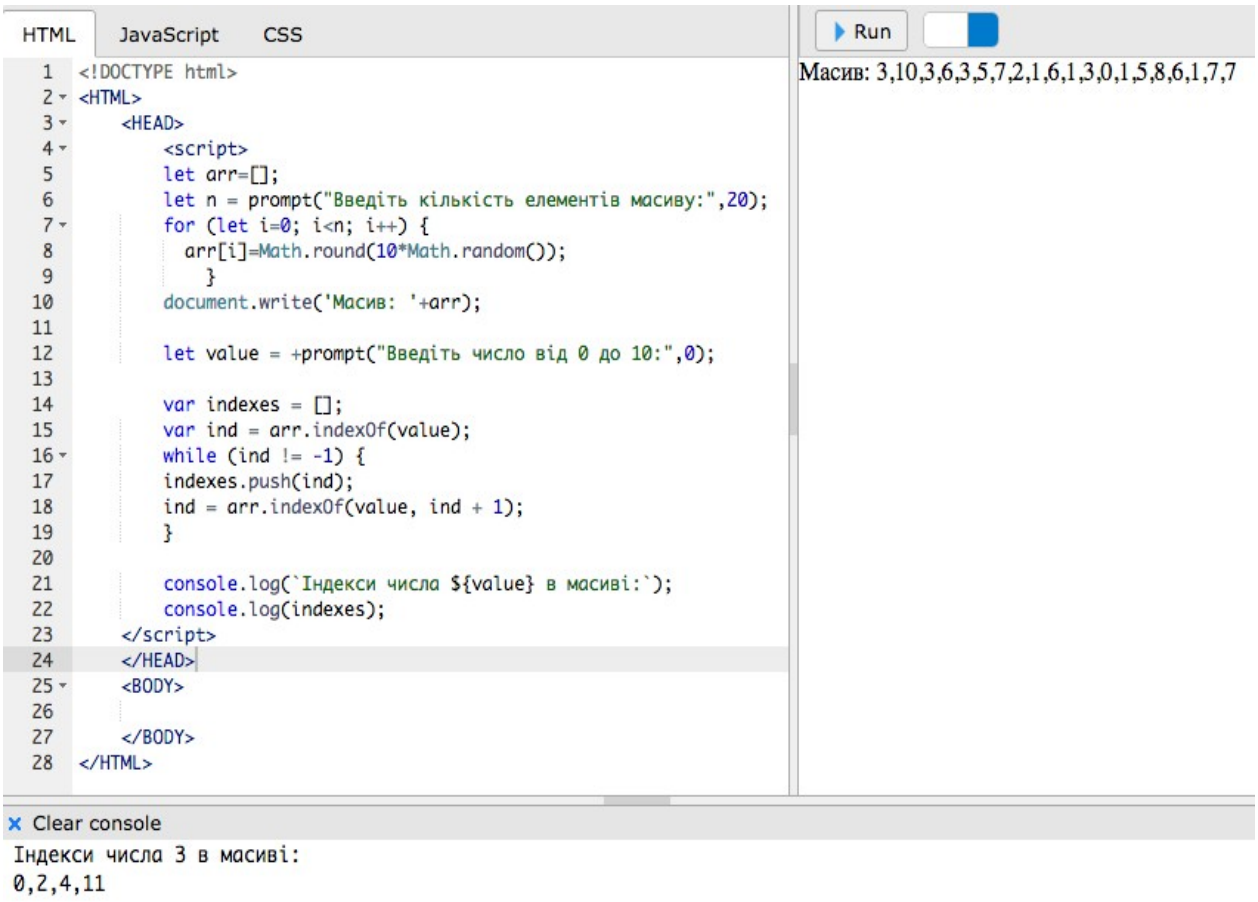


Рис. 5.4. Приклад виконання сценарію.

Приклад. Фрагмент гри на вгадування слів за літерами.
Задано масив з кількох слів. Вивести кількість слів у масиві. Вивести випадкове слово з масиву а) у звичайному вигляді; б) у вигляді рядка, де кожний символ замінений на зірочку. Ввести символ. Якщо символ є у слові, замість зірочок на відповідних позиціях у слові записати цей символ. Для діалогу з користувачем використати функції prompt і alert (див. Рис. 5.5).

```
<script>
         let arr=["spring","summer","fall","winter"];
         let len=arr.length;
         alert(`Є масив з  ${len} слів`);
         let ind=Math.round(Math.random()*(len-1));
         let word=arr[ind];
         alert(`Випадкове слово: ${word}`)
         let code=[];
        for (let i=0;i<word.length;i++) {
            code.push("*");
            }
        alert(`Випадкове слово зашифроване:${code.join("")}`);
        let char=prompt("Введіть літеру:","");
        for (let i=0;i<word.length;i++) {
            if (char==word[i]) {
               code[i]=char;
               }
            }
        alert(`Процес розшифровування: ${code.join("")}`);
</script>
```

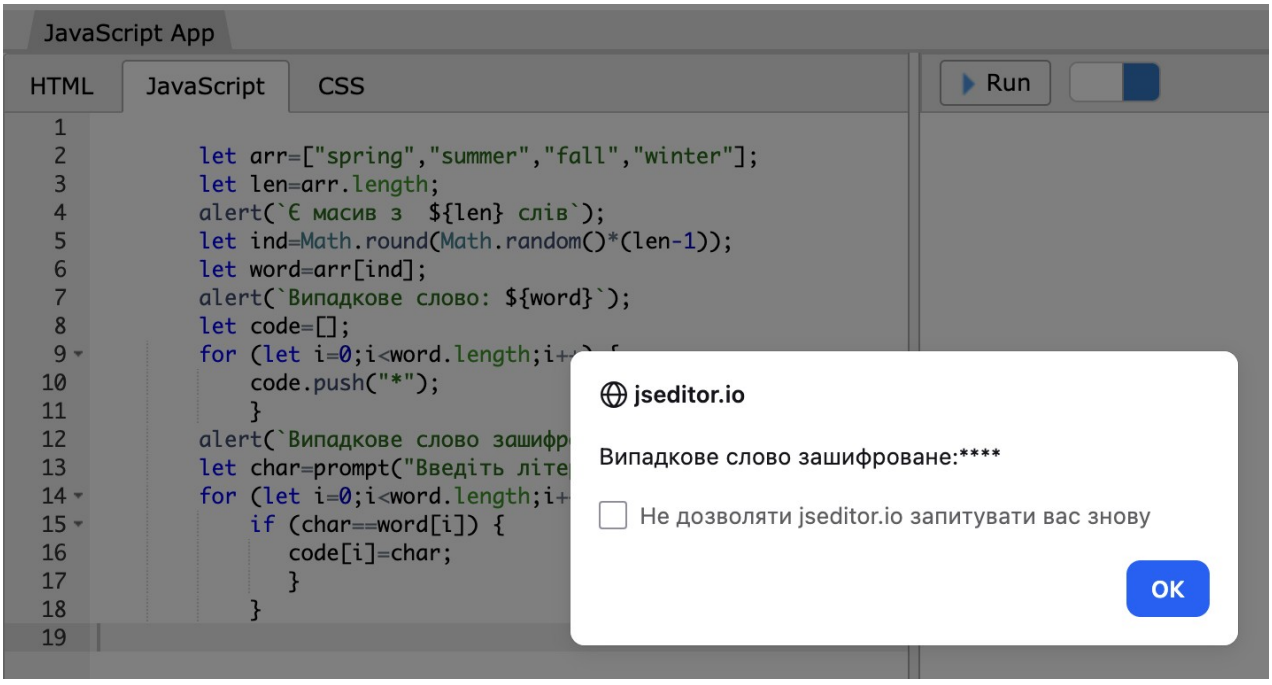


Рис. 5.5. Приклад виконання сценарію.

Об'єкти

Об'єктно-Орієнтоване Програмування (ООП) - один з методів програмування, який розглядає програму як множину «об'єктів», що взаємодіють між собою.

У JavaScript є багато об'єктів, вбудованих у ядро, так званих стандартних об'єктів JavaScript, наприклад: String, Number, Date і т. д.

Кожен стандартний об'єкт є екземпляром об'єкта Object і успадковує його властивості і методи.

Об'єкт у JavaScript - це набір властивостей.

Кожна властивість складається з імені та значення, асоційованого з цим ім'ям.

Значенням властивості може бути функція - ***метод об'єкта.***

На додаток до стандартних об'єктів користувач може визначити свої власні об'єкти. Об'єкти JavaScript, як і в багатьох інших мовах програмування, зазвичай створюються для подання сутностей реального світу, таких як користувачі, замовлення тощо.

Об'єкти в JavaScript поєднують в собі два важливі функціонали:

Перший - це асоціативний масив: структура, що підходить для зберігання довільних даних.

Другий - можливості мови для об'єктно-орієнтованого програмування.

Асоціативний масив - структура даних, в якій можна зберігати дані у форматі ключ-значення.

Ключі властивостей повинні бути рядками, значення можуть бути будь-якого типу.

Способи оголошення об'єкта

Вказується перелік елементів у фігурних дужках (літеральний спосіб):

{ключ1: значення1, ключ2: значення2, ...}

```
let person1 = {'name':'Ivan', 'gender': 'm', 'age':22};
```

Порожній об'єкт подається фігурними дужками без властивостей.

```
let obj = {};
```

Створення об'єкта викликом **new Object()**

let obj = new Object();

```
let person=new Object('name':'Ivan', 'gender': 'm', 'age':22);
```

Основні операції з об'єктами:
створення, отримання і видалення властивостей.

Доступ до властивостей об'єкта здійснюється
за ім'ям властивості (за ключем)

об'єкт[властивість]

через оператор крапка '.' - **об'єкт.властивість**

Приклад. Створення об’єкта для зберігання інформації про людину і отримання відомостей про об’єкт та його окрему властивість (Див. Рис. 5.6):

```
let person2 = {};
console.log('створено порожній об"єкт: '+ person2);
person2.name = 'Vira';
person2.gender = 'f';
person2.age = 20;
console.log('створено об"єкт: '+ person2);
console.log('у об"єкта є властивість name: '+ person2.name);
```

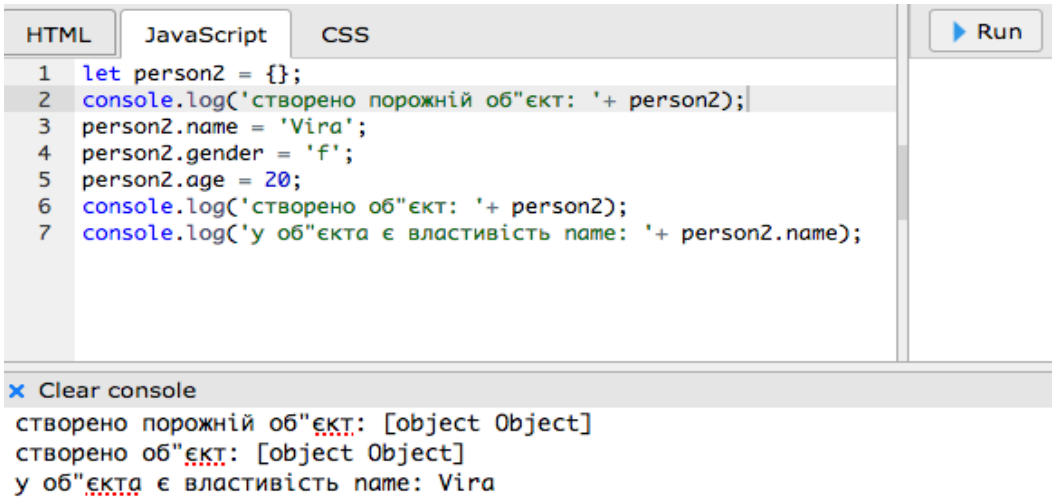


Рис. 5.6. Приклад виконання сценарію.

Додавання властивостей до об’єкта та їх отримання (див. Рис. 5.7):

```
let person2 = {};
person2.name = 'Vira';
person2.gender = 'f';
person2.age = 20;
person2.spec = 'education';
person2.group = 33;
console.log('у об"єкта '+ person2.name + ' є ще такі властивості:'
);
console.log(person2.gender+'    '+person2.age+'    '+person2.spec+'
'+person2.group);
```

Рис. 5.7. Приклад виконання сценарію.

Для видалення властивості використовується оператор delete:
delete об’єкт.властивість

```
delete person2.group; (див. Рис. 5.8)
```

В JavaScript можна звернутися до будь-якої властивості об’єкта, навіть якщо її немає. Якщо властивість не існує, буде повернуто спеціальне значення undefined.

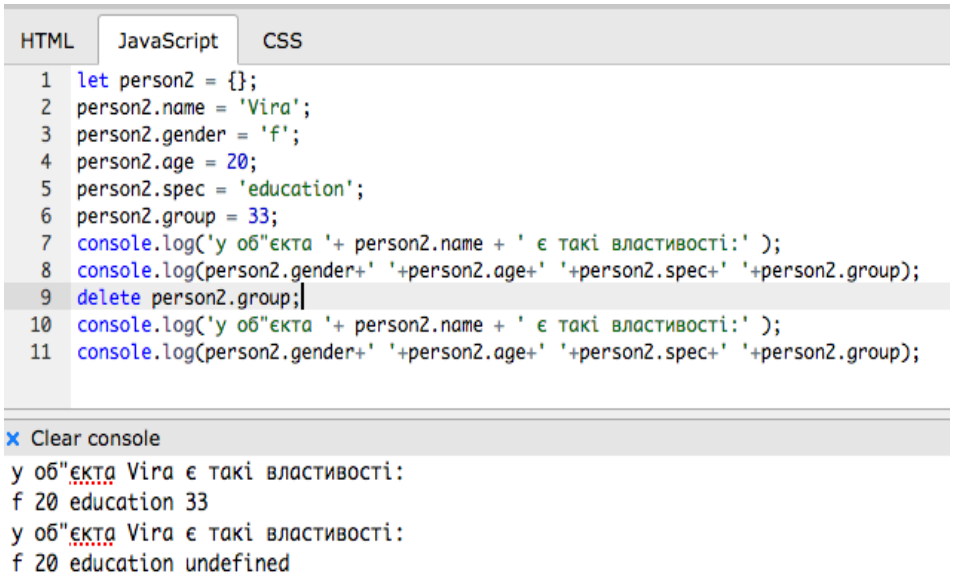


Рис. 5.8. Приклад виконання сценарію.

Перевірка, чи є в об'єкта властивість з потрібним ключем - оператор in.
"властивість" in об’єкт

Ім'я властивості вказується у вигляді рядка (в лапках)

```
if ("group" in person2) {
   alert("Властивість group присутня!");
} (див. Рис. 5.9)
```

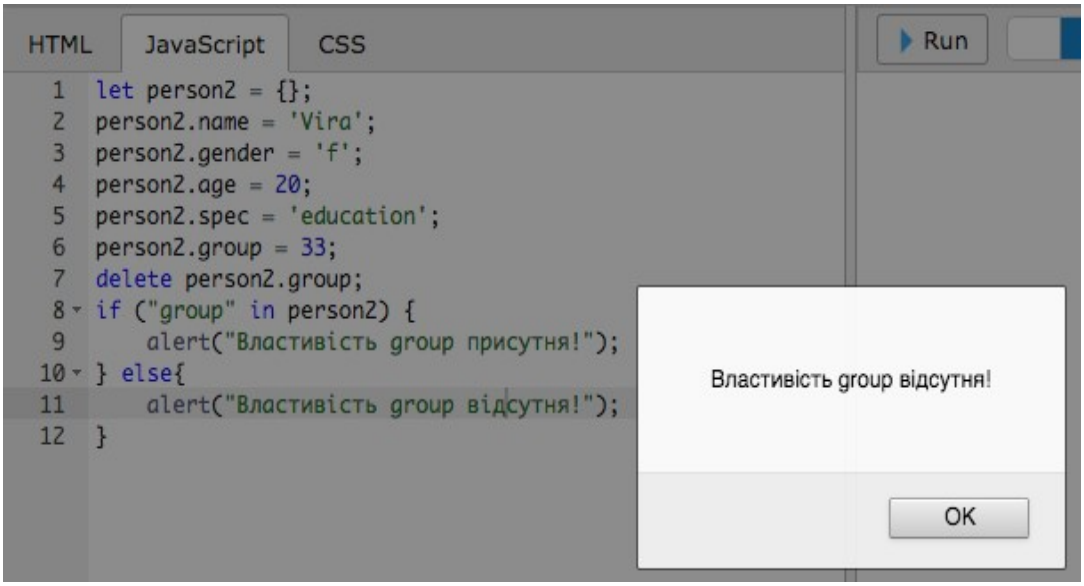


Рис. 5.9. Приклад виконання сценарію.

Для перевірки наявності властивості використовується і порівняння значення з undefined (див. Рис. 5.10)

Рис. 5.10. Приклад виконання сценарію.

Доступ до об'єкта через квадратні дужки

об'єкт['властивість']

Квадратні дужки дають можливість використовувати в якості імені властивості будь-який рядок (із пропусками):

```
person2['last name'] = 'Vovk'; / правильно
person.'last name' = 'Vovk'; // помилка
```

Ім'я властивості має бути рядком. Значення іншого типу приведеться до рядка автоматично.

Доступ до властивості об'єкта через змінну

Якщо ім'я властивості зберігається у змінній, то до нього можна звернутися через квадратні дужки - person2[key].

```
let key = 'spec';
console.log( person2[key] ); // education
```

Доступ через крапку використовується, якщо на етапі написання програми відома назва властивості. Якщо вона визначається у процесі виконання (наприклад: вводиться користувачем і записується в змінну), то використовуються квадратні дужки.

Перебір всіх властивостей об’єкта: цикл for..in.

for (let key in object) {
оператор - тіло циклу (виконується для кожної властивості об’єкта)
}

Приклад. Виведення властивостей і їхніх значень (див. Рис. 5.11)

```
for( let el in person2) {console.log(el + ' ' + person2[el]);}
```

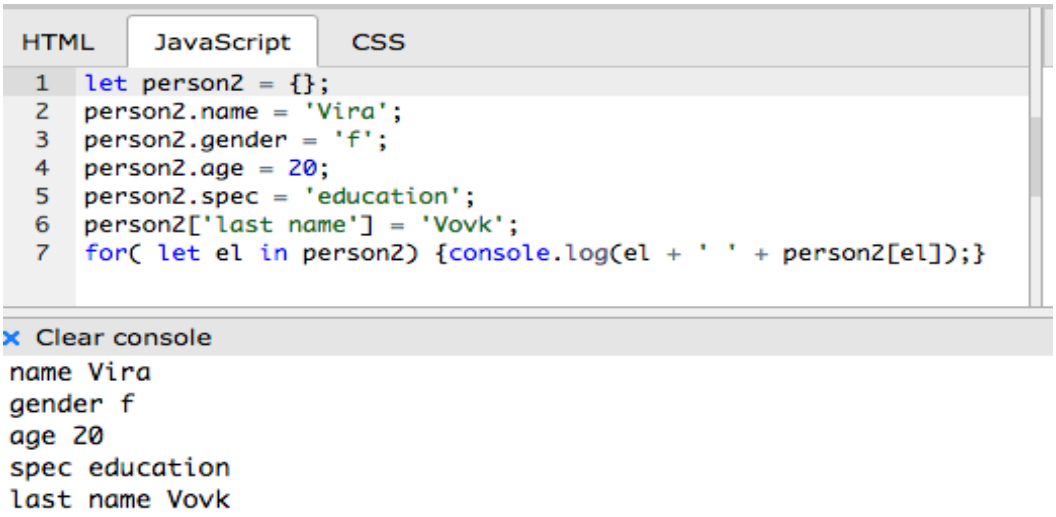


Рис. 5.11. Приклад виконання сценарію.

Властивості об’єкта не обов’язково виводяться у тому порядку, в якому були додані. Властивості з цілочисельними ключами сортуються за зростанням, інші розташовуються в порядку створення.

Цілочисельна властивість - рядок, який може бути перетворений в ціле число і назад без змін, наприклад, "303". "+303" або "30.3" не є цілочисельними властивостями.

Визначення методів

Об’єкти подають сутності реального світу. В реальному світі сутність може діяти. Дії представлені в JavaScript функціями у властивостях об’єкта.

Функція, яка є властивістю об’єкта, називається його ***методом***.

Метод - це функція, асоційована з об’єктом (властивість об’єкта, що є функцією). Методи визначаються так само, як звичайні функції, але вони присвоюються властивості об’єкта.

Приклад. Використання функціонального виразу для створення функції та її присвоєння властивості об’єкта person2.sayHi: (див. Рис. 5.12)

```
let person2 = {name:'Vira',gender:'f'};
person2.sayHi = function() {
    alert("Hello!");
};
person2.sayHi();
```

Рис. 5.12. Приклад виконання сценарію.

Приклад. Використання в якості метода попередньо оголошеної функції sayHi:(Рис. 5.13)

```
let person2 = {name:'Vira',gender:'f'};
function sayHi() {
    alert("Hello!");
}
person2.sayHi=sayHi;
person2.sayHi();
```

Рис. 5.13. Приклад виконання сценарію.

Матеріали для самоосвіти:
Стандартні об'єкти javascript
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects
Порівняння Object і Map
https://webdoky.org/uk/docs/Web/JavaScript/Reference/Global_Objects/Map/#opys

6. ОБ'ЄКТНА МОДЕЛЬ ДОКУМЕНТА DOM (DOCUMENT OBJECT MODEL)

Структура браузерних об’єктів

JavaScript працює у веб браузері в контексті такої структури (Рис. 6.1):

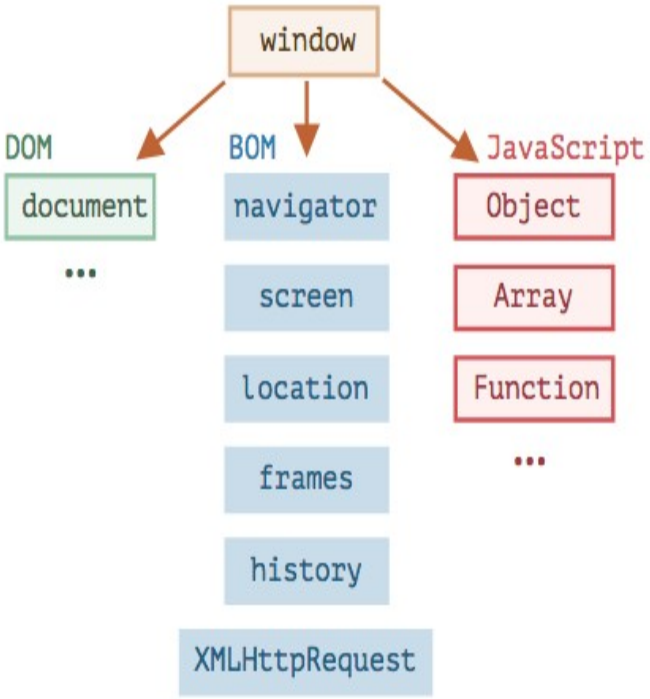


Рис. 6.1. Структура браузерних об’єктів

Об’єкт window

На вершині структури розташовується window.

Цей об'єкт з одного боку є глобальним об'єктом в JavaScript, з іншого - містить властивості та методи для управління вікном браузера, відкриття нових вікон.

Приклад:

```
window.open('http://chnpu.edu.ua'); // відкрити нове вікно
```

Browser object model (BOM)

BOM - це додаткові об’єкти, надані браузером (хост-середовищем) для роботи з усім, крім документа.

Наприклад: об'єкт **navigator** містить загальну інформацію про браузер і операційну систему.

navigator.userAgent - містить інформацію про поточний браузер.

navigator.platform - містить інформацію про платформу, на якій відкрито браузер – Windows/Linux/Mac тощо).

Об'єкт **location** містить інформацію про поточний URL сторінки і дозволяє перенаправити відвідувача на новий URL.

Функції alert, confirm і prompt теж входять в BOM. Вони безпосередньо не пов’язані з документом, але є методами взаємодії з користувачем.

Приклад використання:

```
alert(location.href); // вивести поточну адресу (див. Рис. 6.2)
```

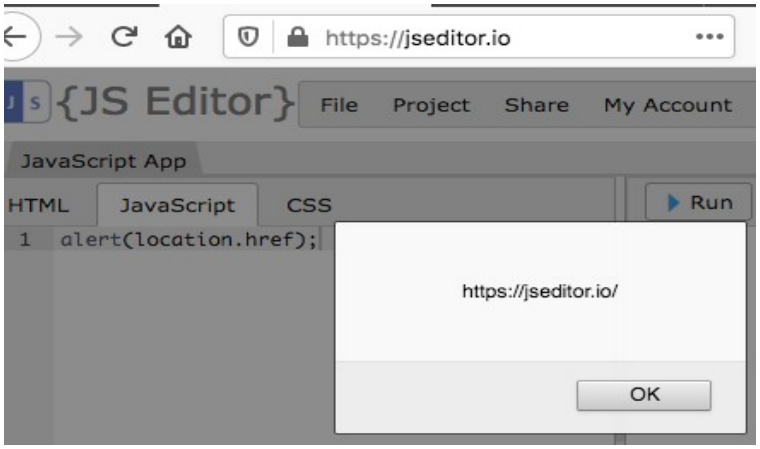


Рис. 6.2. Приклад виконання сценарію.

Об'єктна модель документа **(Document Object Model, DOM**) - це засіб для роботи зі структурою документа, а також з елементами сторінки в кодах HTML та у сценаріях.

Основні аспекти структури і функціонування DOM:

- DOM - подання вебсторінки, схоже на вихідний код HTML, але відмінне від нього.
- Для завантаження вебсторінки браузер за вказаною URL адресою відправляє запит і завантажує з сервера вебсторінку у вигляді HTML коду (вихідного коду сторінки). Якщо у коді вказані інші файли - css, js - вони також завантажуються.
- Із завантаженого з сервера HTML коду браузер формує DOM.
- DOM має деревоподібну структуру і складається з вузлів.
- Кожен вузол DOM формується з HTML тегу і отримує властивості, події, стилі, які вказані в атрибутах тегу, CSS стилях і в JavaScript коді.
- DOM підтримує об'єктно орієнтоване представлення вебсторінки і дозволяє змінювати документ вебсторінки за допомогою JavaScript.
- DOM формується для того, щоб за допомогою JavaScript можна було маніпулювати веб документом: шукати потрібний елемент, додавати нові елементи, отримати наступний дочірній елемент і т.п.
- У JavaScript для роботи з DOM є об'єкт document, який містить методи і властивості для роботи з документом.
- Вигляд DOM документа зображується у панелі розробника в браузері (див. Рис.6.3).

На Рис. 6.4. зображена деревоподібна структура DOM веб документа.

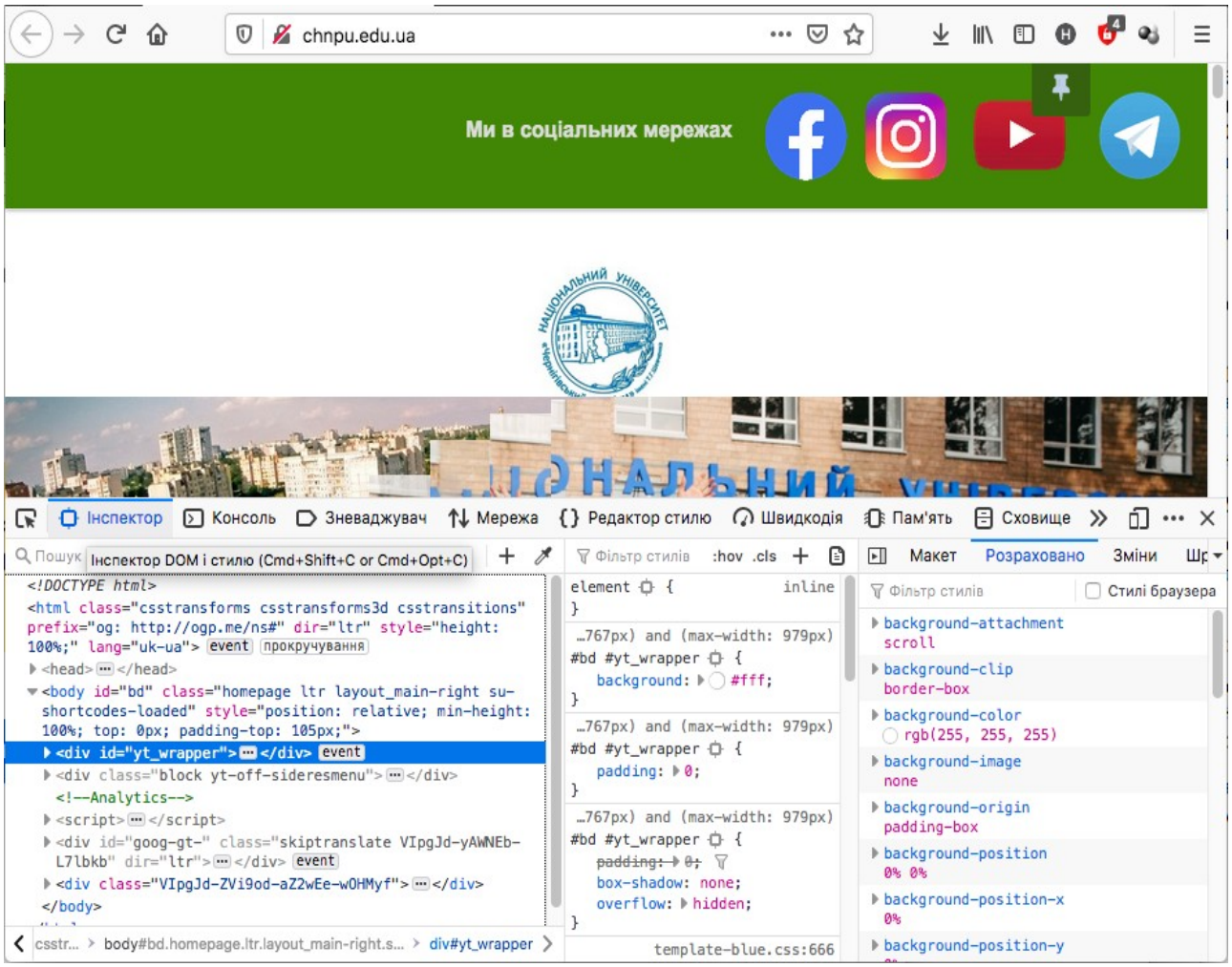


Рис. 6.3. Зображення DOM документа у панелі розробника в браузері.

```
<html>
<head>
  <title>Заголовок веб-сторінки</title>
</head>
<body>
  <p>Абзац 1</p>
  <div>
    <p>Абзац 2</p>
    <p>Абзац 3</p>
    <div>
      <b>Жирний текст</b>
      <a href="http://яваскрипт.укр/">JavaScript</a>
    </div>
  </div>
</body>
</html>
```

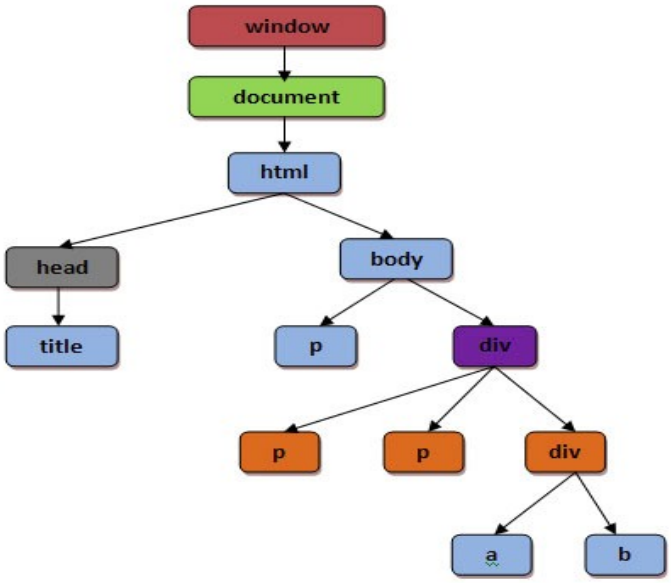


Рис. 6.4. Деревоподібна структура DOM веб документа.

Теги – це основа HTML-документа.

Відповідно до об'єктної моделі документа (DOM), кожен HTML-тег є об'єктом. Вкладені теги – це "діти" всередині батьківського елементу. Текст всередині тегу також є об'єктом.

Всі ці об'єкти доступні за допомогою JavaScript, і їх можна використовувати, щоб змінити сторінку.

Приклад.

document.body - це об'єкт, що представляє тег <body>.

```
document.body.style.background = "red";// змінити колір фону на червоний
setTimeout(() => document.body.style.background = "", 3000);// повернути його назад після 3 секунд
```

Приклад DOM

Є документ

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Про лосів</title>
</head>
<body>
  Правда про лосів.
</body>
</html>
```

DOM подає HTML як структуру дерева тегів (Рис. 6.5):

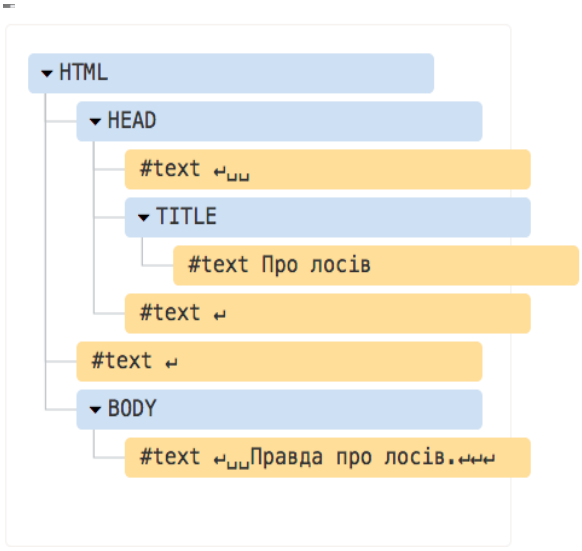


Рис. 6.5. Дерево тегів

Кожен вузол дерева є об'єктом.

Теги є вузлами-елементами (або просто елементами), вони утворюють структуру дерева:

<html> знаходиться в корені,

<head>, <body> – це його дочірні вузли тощо.

Текст всередині елементів утворює текстові вузли, позначені як #text.

Текстовий вузол містить лише рядок. У нього не може бути нащадків, тобто він міститься на найнижчому рівні (є листком дерева).

Наприклад, у тегу <title> текстовий вузол "Про лосів".

В текстових вузлах є спеціальні символи

- новий рядок: ↵ (в JavaScript - \n)
- пробіл: ␣

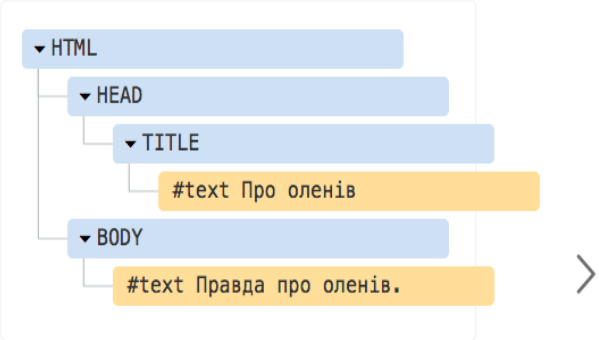
Пробіли та нові рядки є повноправними символами, як літери та цифри. Вони утворюють текстові вузли і стають частиною DOM. Тег <head> містить кілька пробілів перед <title>, котрі утворюють текстовий вузол #text (він містить лише символ нового рядка та декілька пробілів).

Винятки з цього правила:

Пробіли та нові рядки до <head> ігноруються з історичних причин.

Текст після </body> автоматично розміщується всередині body, в кінці, оскільки специфікація HTML вимагає, що весь вміст повинен бути всередині <body>.

В інших випадках будь-які пробіли, як і інші символи, у документі стають текстовими вузлами в дереві DOM, і якщо видалити ці пробіли, то текстових вузлів там не буде.



Приклад.

```
<!DOCTYPE HTML>
<html><head><title>Про
оленів</title></head><body>Правда
про оленів.</body></html>
```

Тут немає текстових вузлів з пробілами (Рис. 6.6):

Рис. 6.6. Дерево тегів

Пробіли у рядку на початку/в кінці і ті вузли, що містять тільки пробіли, як правило, приховані в інструментах розробки, таким чином зберігаючи екранний простір.

Під час створення DOM браузери автоматично обробляють помилки у документі, закривають теги тощо.

Окрім елементів та текстових вузлів є деякі інші типи вузлів, наприклад, коментарі і директива <!DOCTYPE HTML>.

Об’єкт document, який представляє весь документ формально також є вузлом DOM.

Існує 12 типів вузлів. Основні з них:
document – “пункт входу” в DOM.
вузли-елементи – HTML-теги, основні будівельні блоки дерев.
текстові вузли – містять текст.
коментарі – туди можуть бути записані певні відомості, вони не будуть показані, але JS може читати їх з DOM.

Засіб Live Dom Viewer демонструє структуру DOM у режимі реального часу.
http://software.hixie.ch/utilities/js/live-dom-viewer/
Треба вести в спеціальне поле текст документа, і структура буде показана.
Вигляд DOM документа зображується у панелі розробника в браузері.
Матеріали для самоосвіти:
Про роботу з інструментами розробника
https://uk.javascript.info/dom-nodes

Навігація об’єктною моделлю документа

DOM дозволяє маніпулювати HTML- елементами і їхнім вмістом, але для цього слід отримати доступ до потрібного елемента.

Операції з DOM починаються з об'єкта document. З нього можна отримати доступ до будь- яких вузлів.

Вигляд основних посилань, за якими можна переходити між вузлами DOM подано на Рис. 6.7:

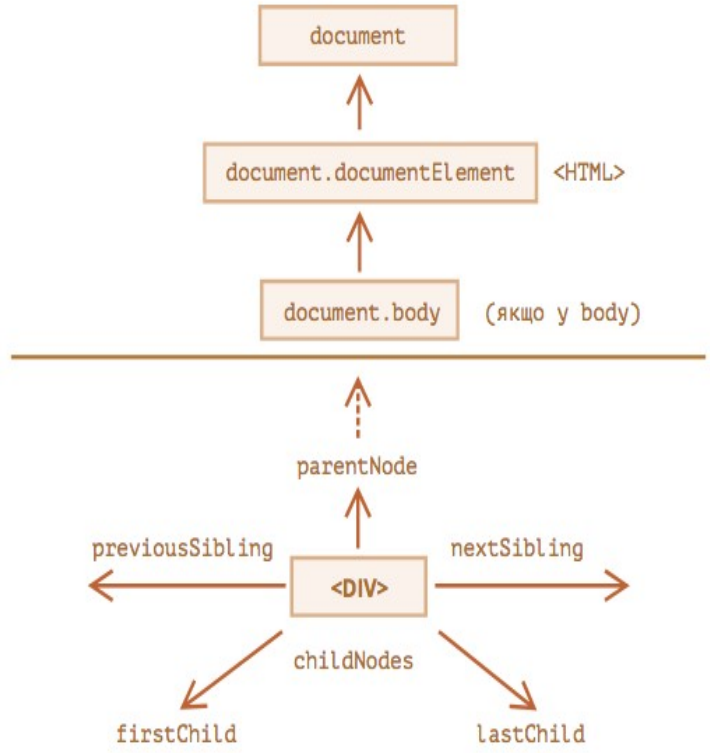


Рис. 6.7. Навігація вузлами DOM.

Верхні елементи

Найвищі елементи дерева доступні як властивості document:

<HTML> = document.documentElement

Самий верхній вузол документа: document.documentElement.

У DOM він відповідає тегу <html>.

<BODY> = document.body

Інший часто вживаний DOM-вузол – вузол тегу <body>: document.body.

<head> = document.head

Тег <head> доступний як document.head.

Приклад.

```
document.body.style.background = 'yellow';
```

Нижче наведено повний код веб документа і приклад виконання сценарію (Рис. 6.8).

```
<!DOCTYPE html>
<html>
<head>
    <style>
    h1 {text-align: center; font-size: 22pt; color: #00ffff;}
    </style>

  <title>Вступ до DOM</title>
</head>
   <body>
     <h1 id="main-heading">Вітання!</h1>
     <p>Текст абзацу</p>
     <script>
      document.body.style.background = 'yellow';
     </script>
   </body>
</html>
```

Рис. 6.8. Приклад виконання сценарію.

Дочірні елементи childNodes, firstChild, lastChild

Дочірні елементи (або діти) - елементи, що лежать безпосередньо всередині даного.

Наприклад, всередині <HTML> зазвичай лежать <HEAD> і <BODY>.

Нащадки - все елементи, що лежать всередині даного, разом з їхніми дітьми, дітьми їхніх дітей і так далі. Тобто, все піддерево DOM.

childNodes

Псевдо-масив childNodes зберігає всі дочірні елементи, включаючи текстові.

Приклад. Послідовне виведення всіх дочірніх елементів document.body:

```
for (let i = 0; i < document.body.childNodes.length; i++)
 {
 alert( document.body.childNodes[i] );
}
```

Нижче наведено повний код веб документа і приклад виконання сценарію (Рис. 6.9).

```
<!DOCTYPE html>
<html>
<head>
    <style>
    h1 {text-align: center; font-size: 22pt; color: #00ffff;}
    </style>

  <title>Вступ до DOM</title>
</head>
   <body>
     <h1 id="main-heading">Вітання!</h1>
     <p>Текст абзацу</p>
     <script>
      let childs = document.body.childNodes.length;
      for (let i = 0; i < childs; i++) {
        (alert(document.body.childNodes[i]));
      }
     </script>
   </body>
</html>
```

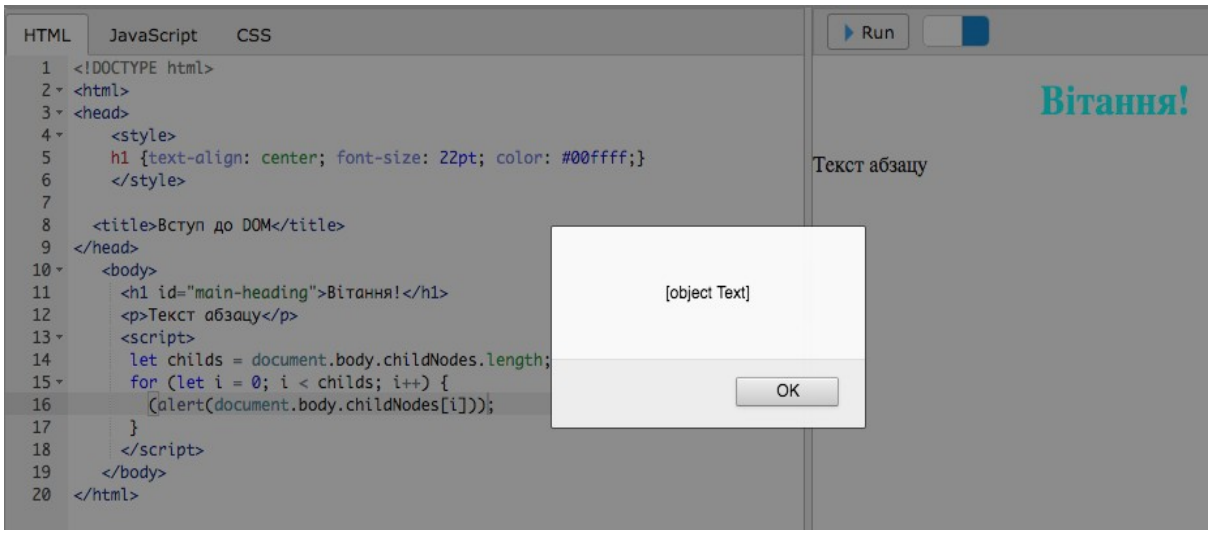


Рис. 6.9. Приклад виконання сценарію.

firstChild і ***lastChild***

Властивості firstChild і lastChild забезпечують швидкий доступ до першого і останнього дочірніх вузлів.

При наявності дочірніх вузлів справджується рівність:

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

Сусіди і батько

Сусіди, або сусідні вузли – це вузли, які мають однаковий батьківський елемент.

Наприклад, тут <head> і <body> є сусідами:

```
<html>
  <head>...</head><body>...</body>
</html>
```

<body> вважається “наступним” або сусідом “праворуч” для <head>
<head> вважається “попереднім” або сусідом “ліворуч” для <body>

Доступ до елементів зліва і справа від даного можна отримати за властивостями ***previousSibling*** / ***nextSibling*** (див. код нижче і Рис. 6.10).

```
<!DOCTYPE html>
<html>
<head></head><body><script>
alert( document.body.parentNode === document.documentElement ); // true
alert( document.head.nextSibling); // HTMLBodyElement
alert( document.body.previousSibling); // HTMLHeadElement
</script>
</body>
</html>
```

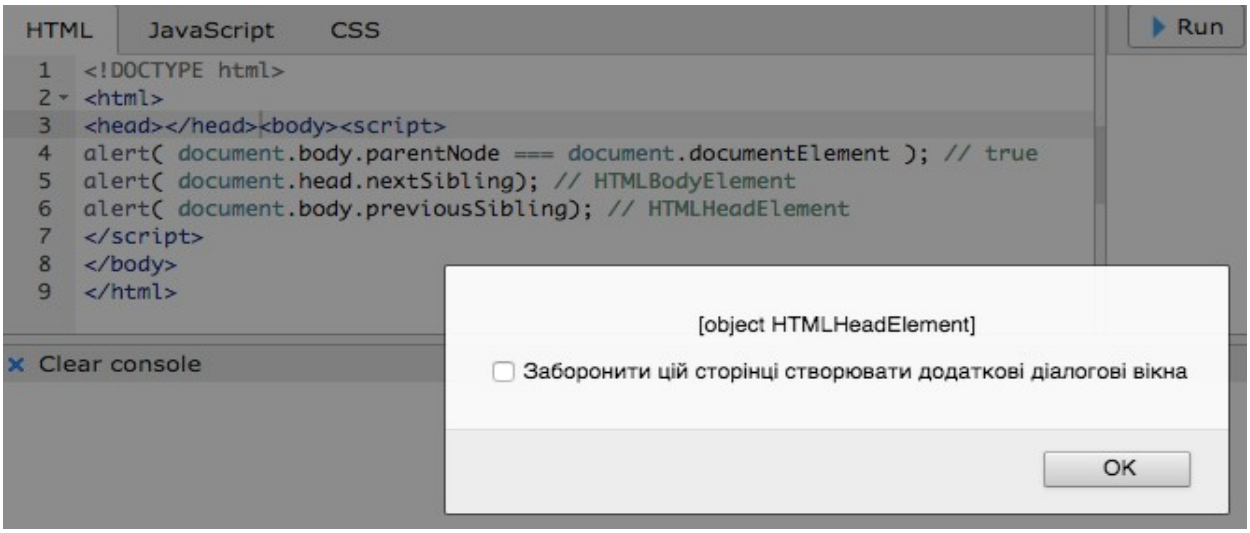


Рис. 6.10. Приклад виконання сценарію.

Батьківський вузол доступний через ***parentNode***.

Приклад.

```
alert( document.body.parentNode === document.documentElement ); // true
```

Навігація тільки за елементами

Властивості навігації, перераховані вище, відносяться до всіх вузлів.

Наприклад, у childNodes можна побачити як текстові вузли, так і вузли елементів і навіть вузли коментарів, якщо вони існують.

Але для багатьох задач текстові вузли чи вузли коментарів не потрібні, натомість треба маніпулювати вузлами елементів, які подають теги та формують структуру сторінки.

Розглянемо додатковий ***набір посилань, які враховують лише вузли-елементи*** (Рис. 6.11):

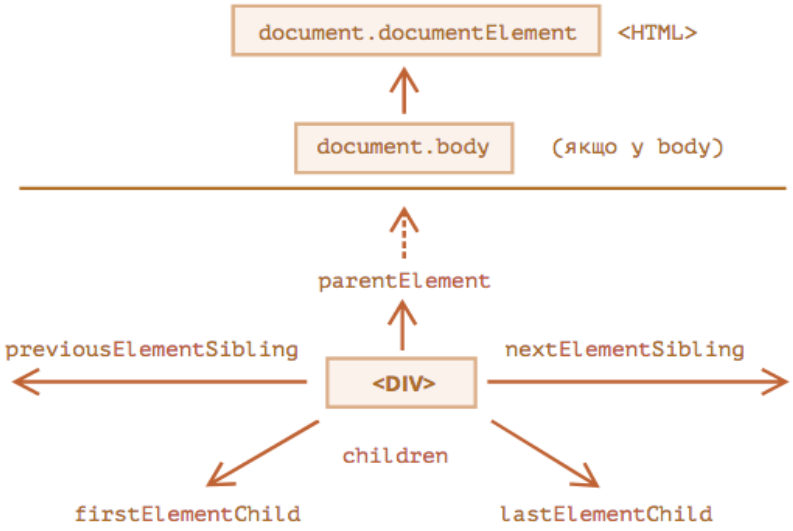


Рис. 6.11. Посилання для вузлів-елементів.

Ці посилання подібні до наведених раніше, але в низці назв є слово *Element:*

children – колекція дітей, котрі є елементами.

firstElementChild, lastElementChild – перший і останній дочірній елементи.

previousElementSibling, nextElementSibling – сусіди-елементи.

parentElement – батько-елемент.

Приклад. Послідовне виведення дочірніх елементів document.body (лише елементів) (Рис. 6.12)

```
<!DOCTYPE html>
<html>
<head>
    <style>
    h1 {text-align: center; font-size: 22pt; color: #00ffff;}
    </style>

  <title>Вступ до DOM</title>
</head>
   <body>
     <h1 id="main-heading">Вітання!</h1>
     <p>Текст абзацу</p>
     <script>
       for (let element of document.body.children) {
           alert(element);
      }
     </script>
   </body>
</html>
```

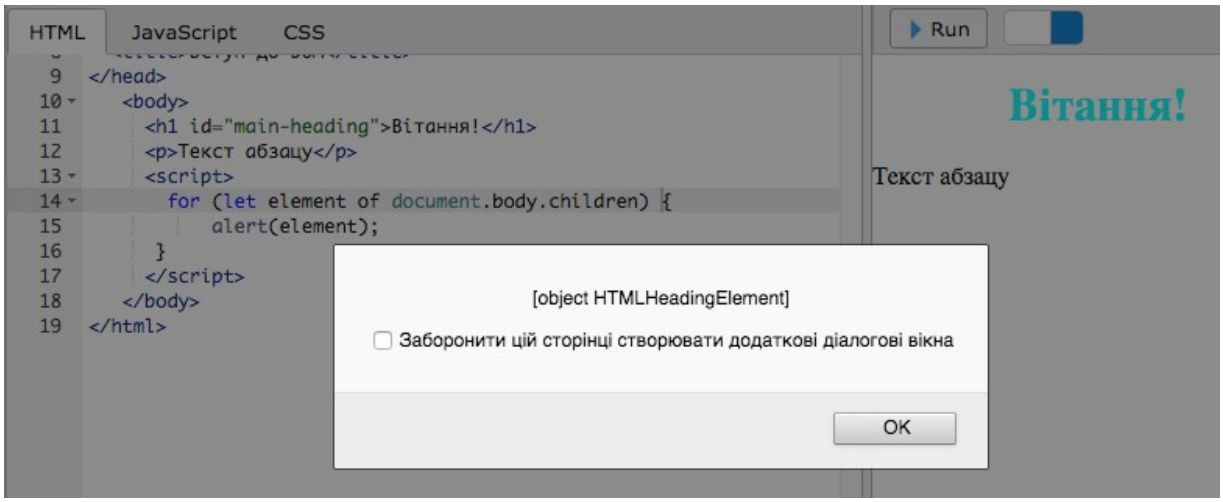


Рис. 6.12. Приклад виконання сценарію.

У конкретних елементів DOM можуть бути свої додаткові посилання для більшої зручності навігації.

Приклад. Навігація елементами таблиці
table.rows - колекція рядків <TR> таблиці.
table.caption / tHead / tFoot - посилання на елементи таблиці table.tBodies - колекція елементів таблиці <TBODY>, їх може бути кілька.
tbody.rows - колекція рядків <TR> секції.
tr.cells - колекція комірок всередині рядка <TR>
tr.sectionRowIndex - номер рядка в поточній секції tr.rowIndex - номер рядка в таблиці
td.cellIndex - номер комірки в рядку

Таким чином, для вузла DOM можна перейти до його безпосередніх сусідів за допомогою властивостей навігації.

Існує два основних набори:

- *Для всіх вузлів:* parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling.
- *Лише для вузлів-елементів*: parentElement, children, firstElementChild, lastElementChild, previousElementSibling, nextElementSibling.

Деякі типи елементів DOM (наприклад, таблиці) надають додаткові властивості та колекції для доступу до їх вмісту.

Пошук у DOM

Таблиця 6.1

Основні методи пошуку елементів у DOM

Метод	**Шукає за...**	**Шукає всередині елемента?**	**Підтримка**
getElementByID	id	-	всюди
getElementsByName	name	-	всюди
getElementsByTagName	тег або '*'	+	всюди
gerElementsByClassName	класом	+	крім IE8
querySelector	CSS-селектором	+	всюди
querySelectorAll	CSS-селектором	+	всюди

Матеріали для самоосвіти:
Довідка про елементи:
http://яваскрипт.укр/Element

Ідентифікація елементів за id

HTML-елементу можна надати унікальне ім'я, або ідентифікатор, за допомогою атрибута **id.**
Наприклад, елемент h1 має атрибут id:

```
<h1 id="main-heading">Вітання!</h1>
```

За цим id можна згодом знайти цей конкретний заголовок і використати у коді, без розгляду інших елементів, у тому числі заголовків рівня h1

Пошук елемента за допомогою getElementById

Загальноприйнятою практикою є доступ до елемента викликом
`document.getElementById("ідентифікатор")`

Позначивши елемент унікальним id у межах документа, можна скористатися DOM-методом document.getElementById, щоб отримати доступ до вказаного елемента, наприклад:

```
var headingElement = document.getElementById("main-heading");
```

Глобальний об'єкт document дозволяє взаємодіяти із вмістом сторінки.

Виклик document.getElementById("main-heading") дає браузеру команду знайти елемент, id якого дорівнює "main-heading".

Коли елемент знайдено, ним можна керувати за допомогою JavaScript-коду.

Властивість innerHTML

Дозволяє отримати HTML-вміст елемента у вигляді рядка. У innerHTML можна і читати і писати.

Властивість data

Визначає зміст текстового вузла

Властивість innerHTML є тільки у вузлів-елементів.

Доступ до вмісту інших вузлів, наприклад, текстових або коментарів, для читання і запису здійснюється через властивість data. Його теж можна читати і оновлювати.

Властивість textContent

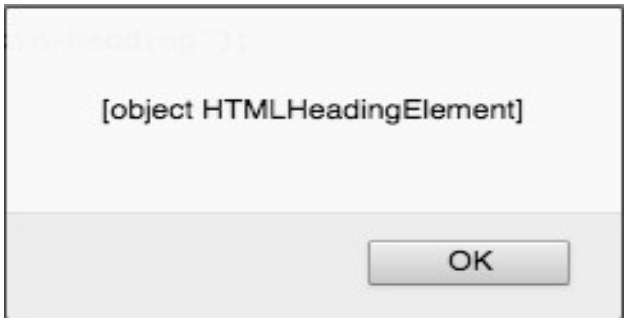
Властивість textContent містить тільки текст всередині елемента, за вирахуванням всіх <тегів>.

Приклад. Через властивість innerHTML можна дізнатися, який текст міститься всередині елемента headingElement, або замінити цей текст:

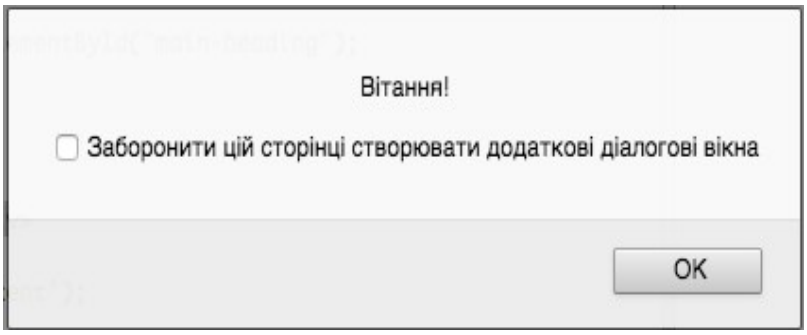
```
headingElement.innerHTML;
```

Ця команда повертає вміст headingElement. В даному випадку вміст - це текст "Вітання!", що знаходиться між тегами <h1> (Див. Рис.6.13)

```
<body>
   <h1 id="main-heading">Вітання!</h1>
   <script>
            var headingElement = document.getElementById("main-
heading");
       alert(headingElement);
       alert(headingElement.innerHTML);
   </script>
</body>
```



а) Результат виконання alert(headingElement);



b) Результат виконання alert(headingElement.innerHTML);

Рис. 6.13. Приклад виконання сценарію.

Приклад. Зміна елемента з використанням innerHTML (див. Рис. 6.14)

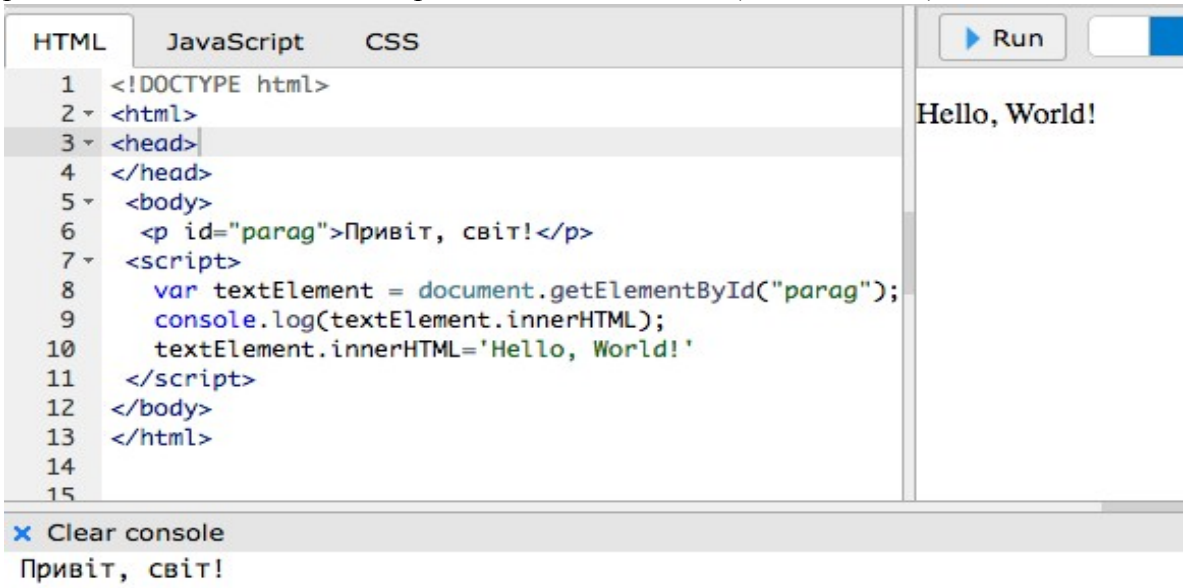


Рис. 6.14. Приклад виконання сценарію.

Приклад. Зміна кольору блоку (Див. Рис.6.15)

```
<!DOCTYPE html>
<html>
<head>
  <title>Вступ до DOM</title>
</head>
<body>
   <h1 id="main-heading">Вітання!</h1>

   <div id="content">Виокремлений елемент</div>
   <script>
     let elem = document.getElementById('content');
     elem.style.background = 'yellow';
   </script>
</body>
</html>
```

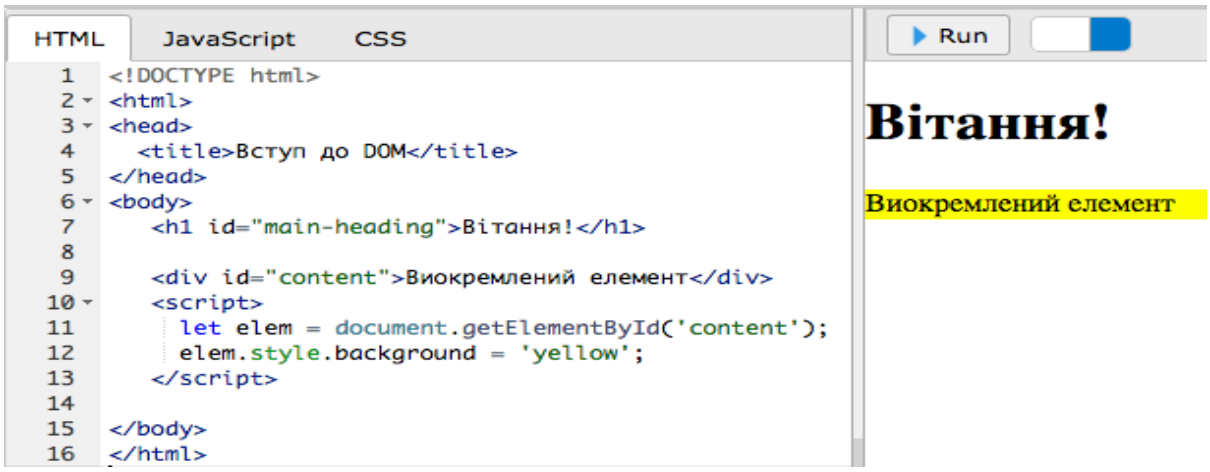


Рис. 6.15. Приклад виконання сценарію.

Приклад. Зміна заголовку (Див. Рис. 6.16)

```
<!DOCTYPE html>
<html>
<head>
  <title>Вступ до DOM</title>
</head>
<body>
   <h1 id="main-heading">Вітання!</h1>
   <script>
    let headingElement = document.getElementById("main-heading");
    console.log(headingElement.innerHTML);
    let newHeadingText = prompt("Введіть новий заголовок:");
    headingElement.innerHTML =newHeadingText;
   </script>
</body>
</html>
```

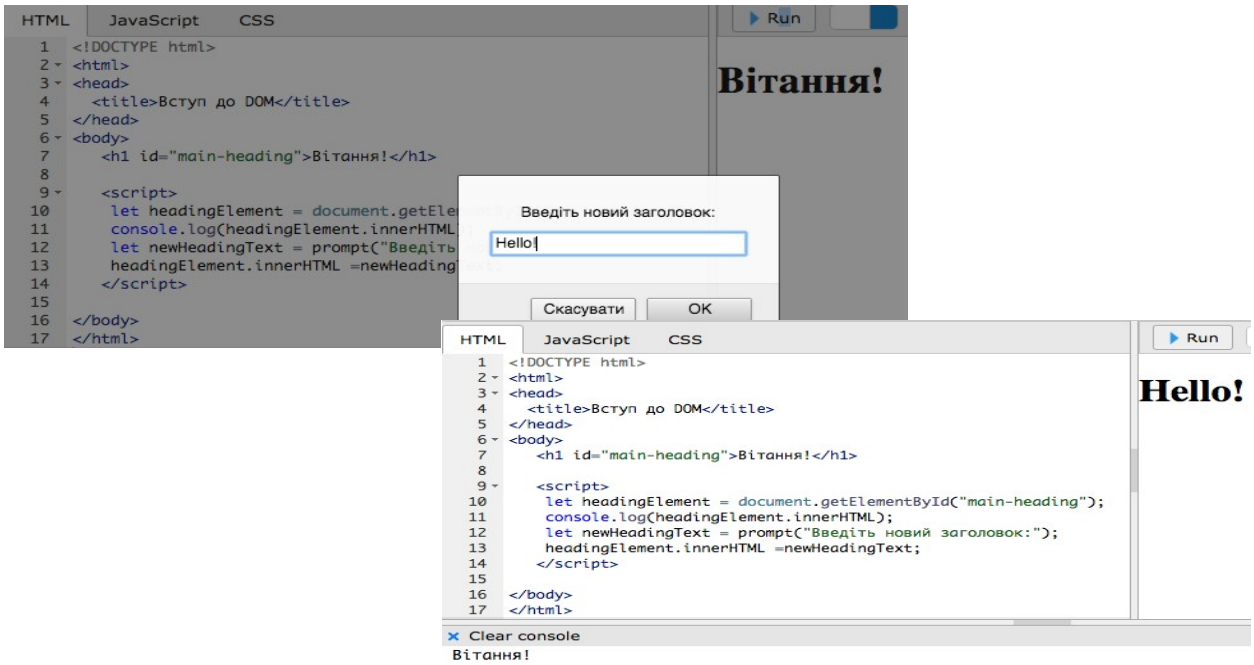


Рис. 6.16. Приклад виконання сценарію.

getElementsByTagName

Метод **`elem.getElementsByTagName(tag)`** шукає всі елементи із заданим тегом tag всередині елемента elem і повертає їх у вигляді списку.

Регістр тегу не має значення.

Приклад:

```
// отримати всі div-елементи
var elements = document.getElementsByTagName('div');
```

На відміну від getElementById, який існує лише в контексті document, метод getElementsByTagName може шукати всередині будь-якого елемента.

Приклад. Вивести всі теги <input> в документі (Див. Рис.6.17).

```
<html>
<head>
</head>
<body>
  <table id="age-table">
  <tr>    <td>Ваш вік:</td>
    <td>
      <label>
       <input type="radio" name="age" value="young" checked> до 18
      </label>
      <label>
       <input type="radio" name="age" value="mature"> від 18 до 50
      </label>
      <label>
       <input type="radio" name="age" value="senior" > від 50
      </label>
    </td>  </tr>
</table>
<script>
  var tableElem = document.getElementById('age-table');
  var elements = tableElem.getElementsByTagName('input');
  for (let i = 0; i < elements.length; i++) {
    let input = elements[i];
    alert( input.value + ': ' + input.checked );
  }
</script>
</body>
</html>
```

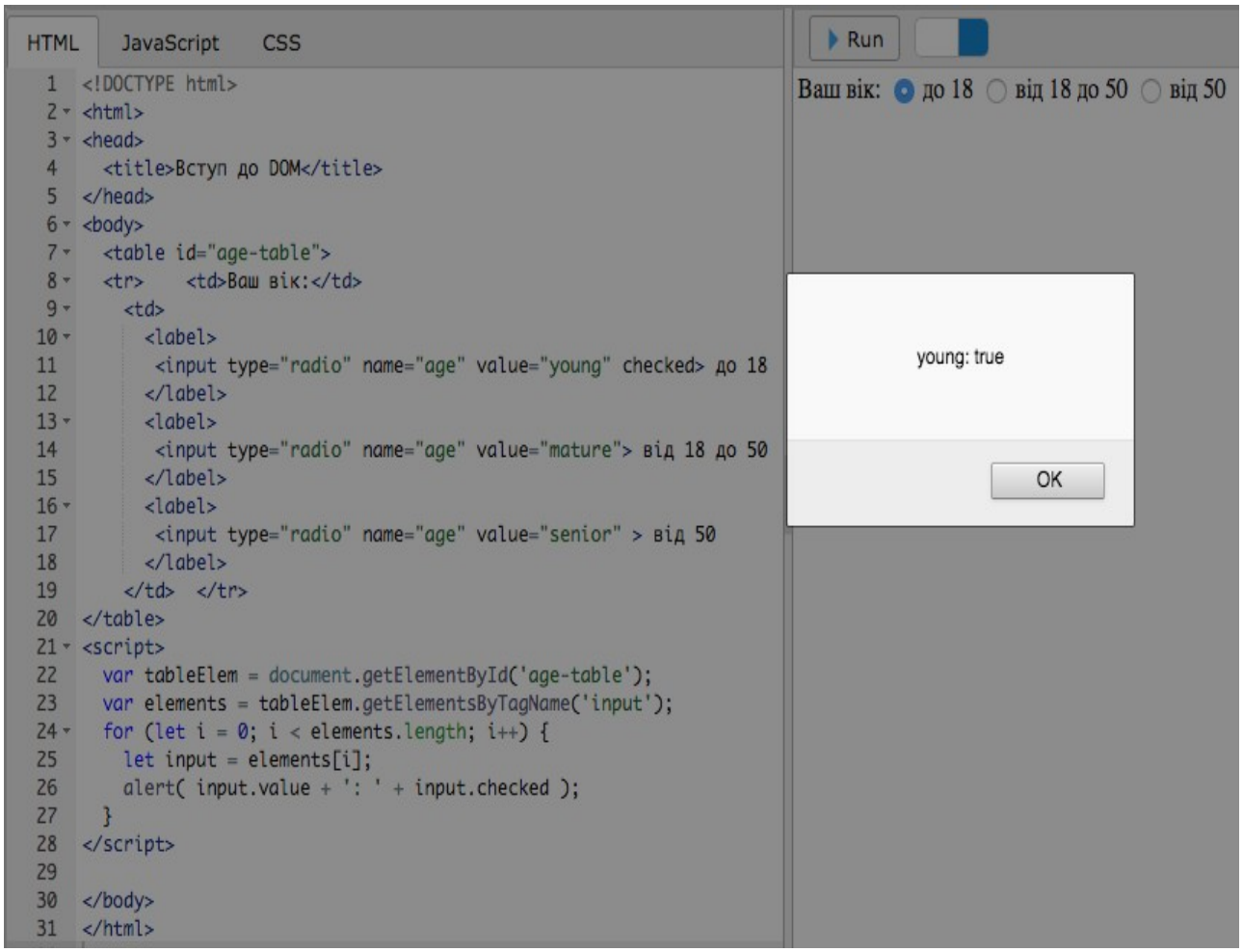


Рис. 6.17. Приклад виконання сценарію.

document.getElementsByName

Виклик **`document.getElementsByName(name)`** дозволяє отримати всі елементи з атрибутом name.

Приклад. Отримати усі елементи з ім'ям age:

```
let elems = document.getElementsByName('age');
```

Приклад. Обчислення кількості елементів “age” в документі (див. Рис. 6.18).

Обчислення кількості елементів “age” в документі

```
<html>
<head>
</head>
<body>
  <table id="age-table">
  <tr>    <td>Ваш вік:</td>
    <td>
      <label>
       <input type="radio" name="age" value="young" checked> до 18
      </label>
      <label>
```

```
       <input type="radio" name="age" value="mature"> від 18 до 50
      </label>
      <label>
       <input type="radio" name="age" value="senior" > від 50
      </label>
    </td>  </tr>
</table>
<script>
  let elems = document.getElementsByName('age');
  alert( "Count of 'age' = " + elems.length);
</script>
</body>
</html>
```

Рис. 6.18. Приклад виконання сценарію.

getElementsByClassName

Виклик **`elem.getElementsByClassName(className)`** повертає колекцію елементів із класом className.

Знаходить елемент у тому випадку, якщо він має кілька класів, а шуканий – один з них.

Як і getElementsByTagName, цей метод може бути викликаний і в контексті DOM-елемента, і в контексті документа.

Приклад. Обчислення кількості елементів з класом “стаття. (Див. Рис. 6.19).

```
<!DOCTYPE html>
<html>
<head>
  <title>Вступ до DOM</title>
</head>
<body>
  <div class="article">Стаття</div>
```

```
  <div class="long article">Довга стаття</div>
  <div class="very long article">Дуже довга стаття</div>
  <div class="extremely long article">Довжелезна стаття</div>
<script>
  let articles = document.getElementsByClassName('article');
  alert( articles.length ); // 4
</script>
</body>
</html>
```

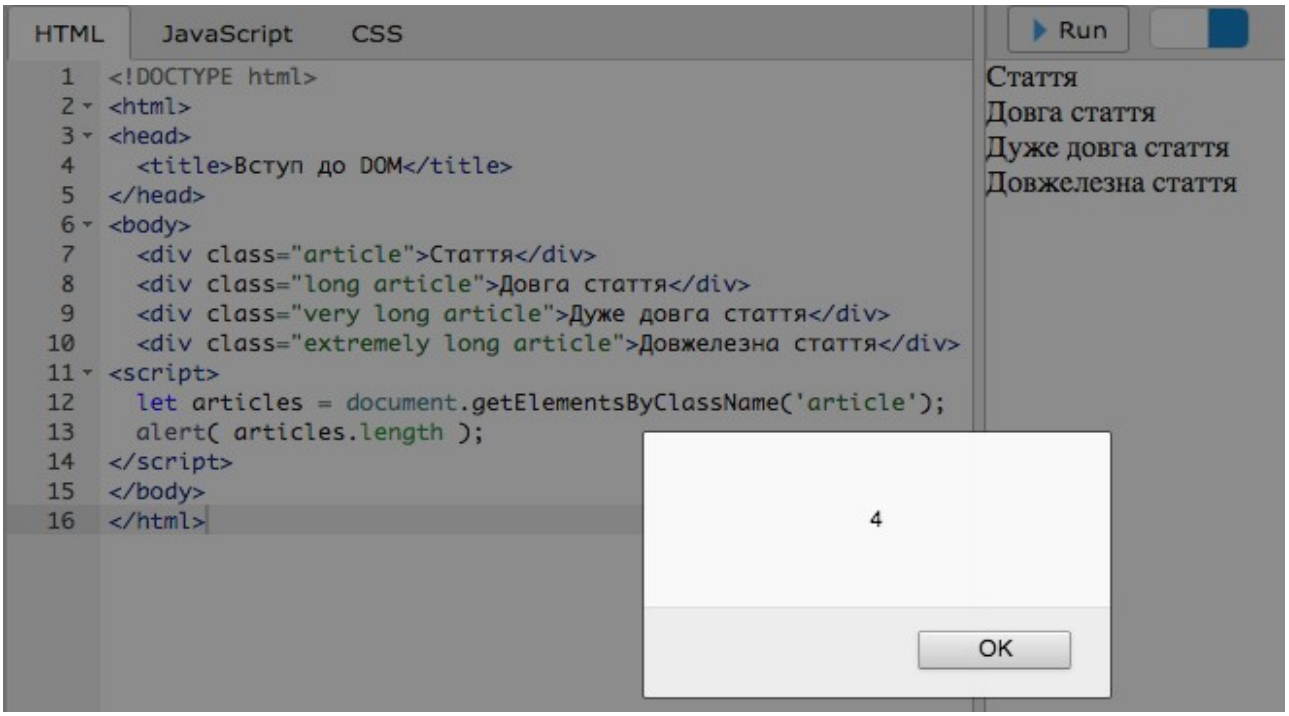


Рис. 6.19. Приклад виконання сценарію.

querySelectorAll

Виклик **`elem.querySelectorAll(css)`** повертає всі елементи всередині elem, які відповідають CSS-селектору css.

Як і getElementsByTagName, цей метод може бути викликаний і в контексті DOM-елемента, і в контексті документа.

Матеріали для самоосвіти:
Приклади селекторів:
http://яваскрипт.укр/CSS%20%D1%81%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D0%BE%D1%80
https://css.in.ua/css/selectors

Приклад. Отримання всіх елементів з класом abc і виведення їхнього вмісту (див. Рис. 6.20).

```
<!DOCTYPE html>
<html>
<head>
   <style>
    .abc {text-align: center; font-size: 12pt; color: #ffa500; }
   </style>
</head>
<body>
 <p class="abc">Це перший абзац. Він має клас abc</p>
 <p class="abc">Це другий абзац. Він теж має клас abc</p>
 <p class="abc">Це третій абзац. І він має клас abc</p>
 <script>
  let elems = document.querySelectorAll('.abc');
 for (let i = 0; i < elems.length; i++) {
    alert( elems[i].innerHTML );
  }
 </script>
</body>
</html>
```

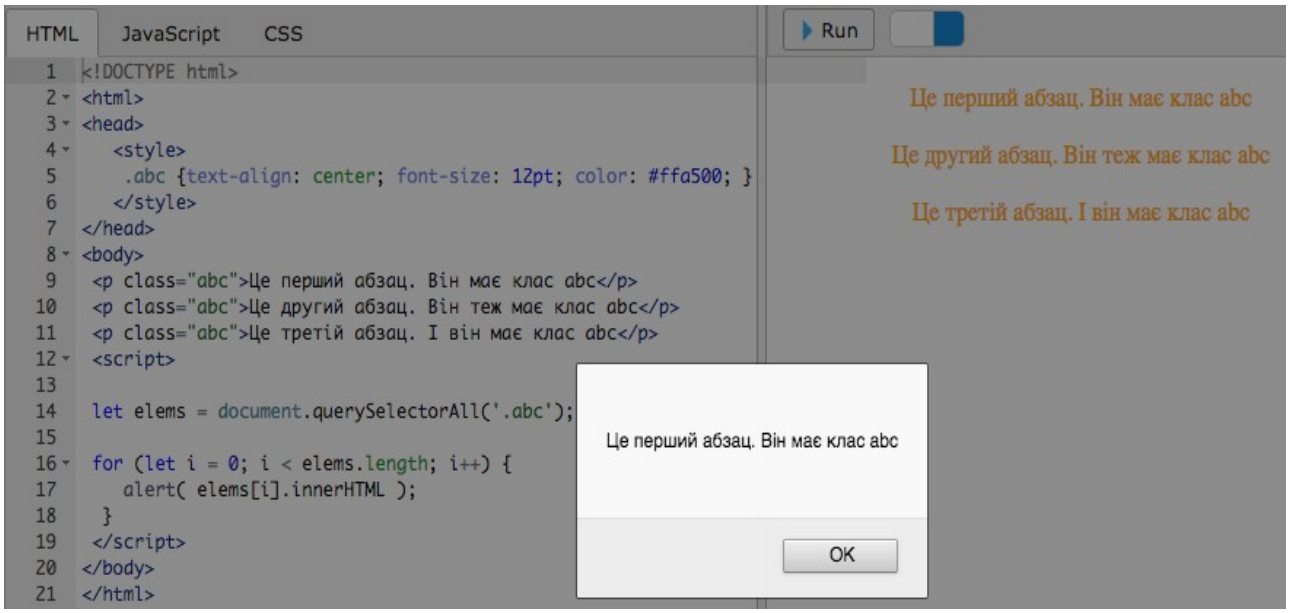


Рис. 6.20. Приклад виконання сценарію.

querySelector

Виклик **`elem.querySelector(css)`** повертає не все, а лише перший елемент, який відповідає CSS-селектору css.

Інакше кажучи, результат – такий самий, як і при elem.querySelectorAll(css)[0], але в останньому виклику спочатку шукаються всі елементи, а потім береться перший, а в elem.querySelector(css) шукається тільки перший, тобто він ефективніше.

Цей метод часто використовується, коли відомо, що відповідний елемент тільки один, і треба отримати в змінну відразу його.

Приклад. Застосування властивості data
Виведення вмісту текстових вузлів та коментарів (Рис. 6.21).

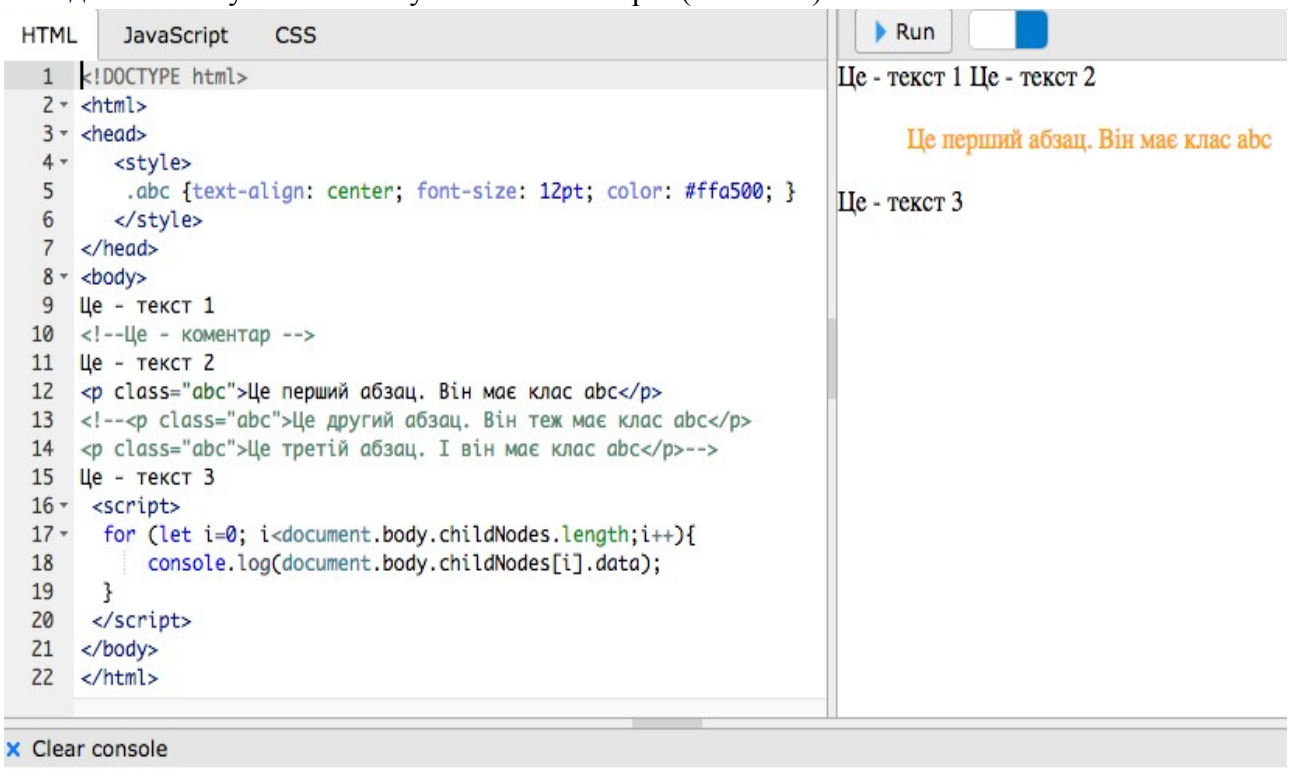


Рис. 6.21. Приклад виконання сценарію.

Властивість ***textContent***

Містить тільки текст всередині елемента, за вирахуванням всіх <тегів>.

Приклад. Виведення тексту всіх *дочірніх вузлів* тіла документа (див. Рис. 6.22).

```
<!DOCTYPE html>
<html>
<head>
   <style>
    .abc {text-align: center; font-size: 12pt; color: #ffa500; }
   </style>
</head>
<body>
Це - текст 1
<!--Це - коментар -->
```

```
Це - текст 2
<p class="abc">Це перший абзац. Він має клас abc</p>
<!--<p class="abc">Це другий абзац. Він теж має клас abc</p>
<p class="abc">Це третій абзац. І він має клас abc</p>-->
Це - текст 3
 <script>
  for (let i=0; i<document.body.childNodes.length;i++){
      console.log(document.body.childNodes[i].textContent);
  }
 </script>
</body>
</html>
```

Рис. 6.22. Приклад виконання сценарію.

Приклад. Виведення тексту всіх *дочірніх елементів* тіла документа (Рис. 6.23).

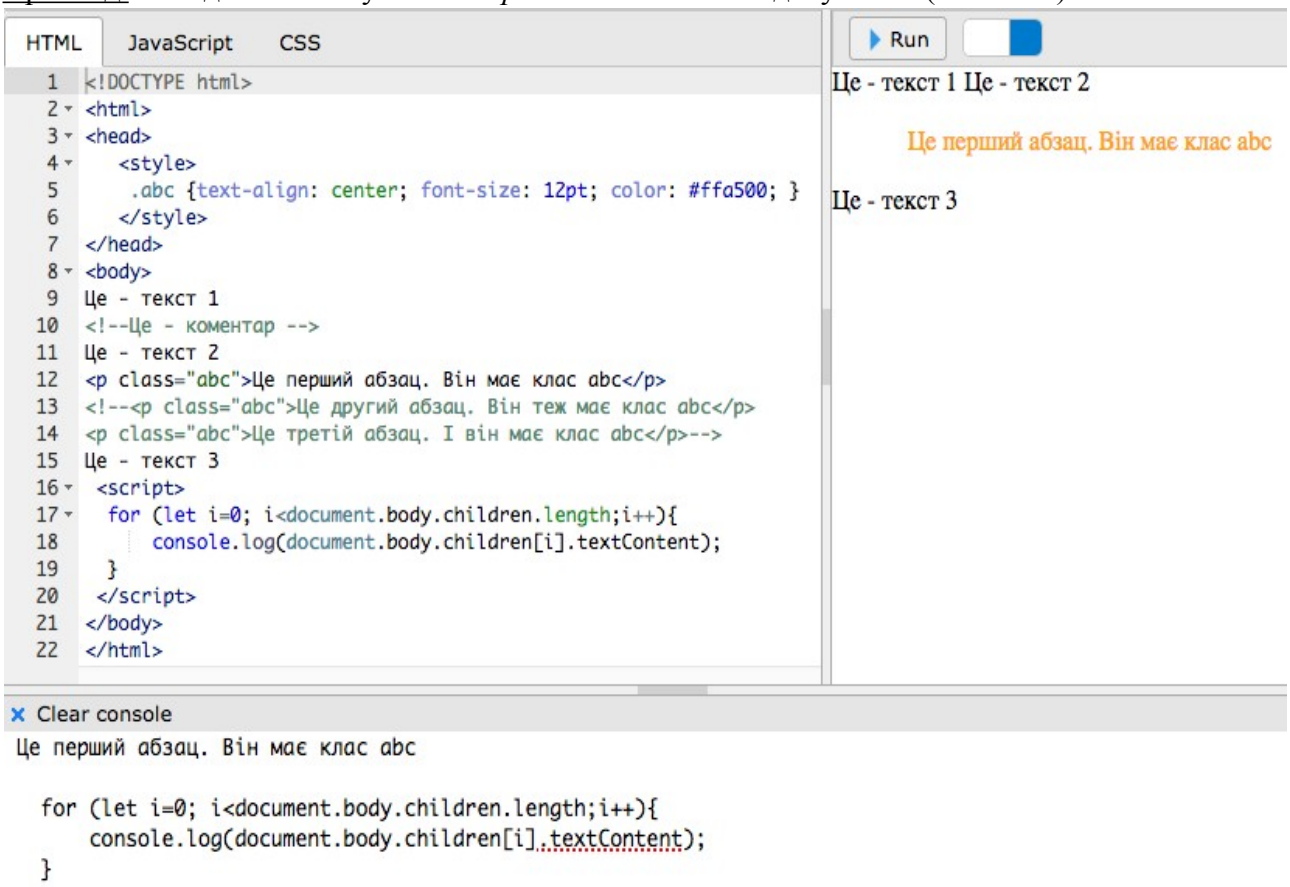


Рис. 6.23. Приклад виконання сценарію.

Документ та стилі елементів

Стилі елемента: властивість style

Властивість **`element.style`** повертає об'єкт, який дає доступ до стилю елемента для читання та запису.

З його допомогою можна змінювати більшість CSS-властивостей.

Приклад. element.style.width="100px" працює так, як задання атрибуту style="width:100px".

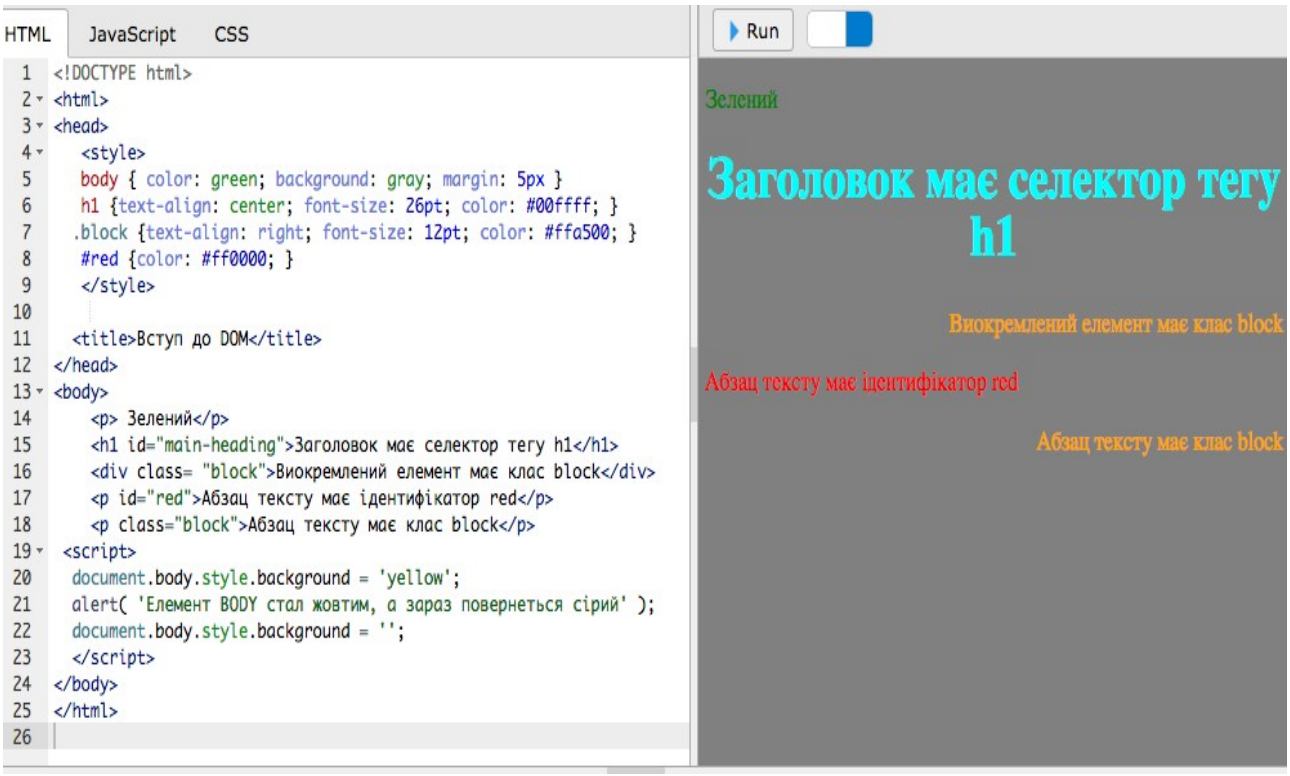
Для властивостей, назви яких складаються з кількох слів, використовується осьТакийЗапис:

```
background-color  => elem.style.backgroundColor
border-left-width => elem.style.borderLeftWidth
```

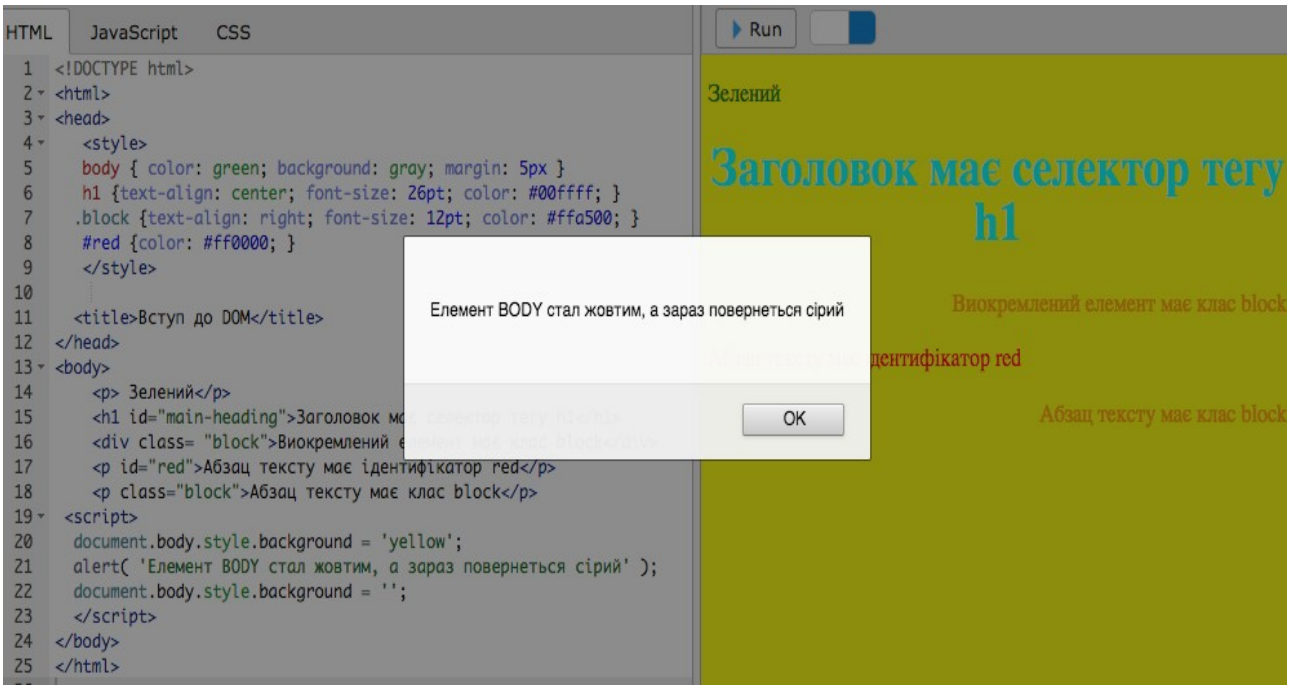
Приклад. Зміна кольору фону документа (див. Рис. 6.24 a), b)).

```
<!DOCTYPE html>
<html>
<head>
  <style>
   body { color: green; background: gray; margin: 5px }
   h1 {text-align: center; font-size: 26pt; color: #00ffff; }
```

```
  .block {text-align: right; font-size: 12pt; color: #ffa500; }
   #red {color: #ff0000; }
  </style>
<title>Вступ до DOM</title>
</head>
<body>
    <p> Зелений</p>
    <h1 id="main-heading">Заголовок має селектор тегу h1</h1>
    <div class= "block">Виокремлений елемент має клас block</div>
    <p id="red">Абзац тексту має ідентифікатор red</p>
    <p class="block">Абзац тексту має клас block</p>
 <script>
  document.body.style.background = 'yellow';
  alert( 'Елемент BODY став жовтим, а зараз повернеться сірий' );
  document.body.style.background = '';
  </script>
</body>
</html>
```



а)



b)Рис. 6.24. Приклад виконання сценарію.

Приклад. Маніпуляції зі стилями (див. Рис.6.25).

```
<!DOCTYPE html>
<html>
<head>
   <style>
   body { color: green; background: lightgray; margin: 5px }
   h1 {text-align: center; font-size: 26pt; color: #ff00ff; }
  .block {text-align: right; font-size: 12pt; color: #ffa500; }
   #red {color: #ff0000; }

   </style>

  <title>Вступ до DOM</title>
</head>
<body>
    <p> Зелений</p>
    <h1 id="main-heading">Заголовок має селектор тегу h1</h1>
    <div class= "block">Виокремлений елемент має клас block</div>
    <p id="red">Абзац тексту має ідентифікатор red</p>
    <p class="block">Абзац тексту має клас block</p>
 <script>
      document.getElementsByTagName("body")[0].style.background   =
"#00bbdd";
  document.getElementsByTagName("div")[0].style.color = "lime";
  </script>
</body>
</html>
```

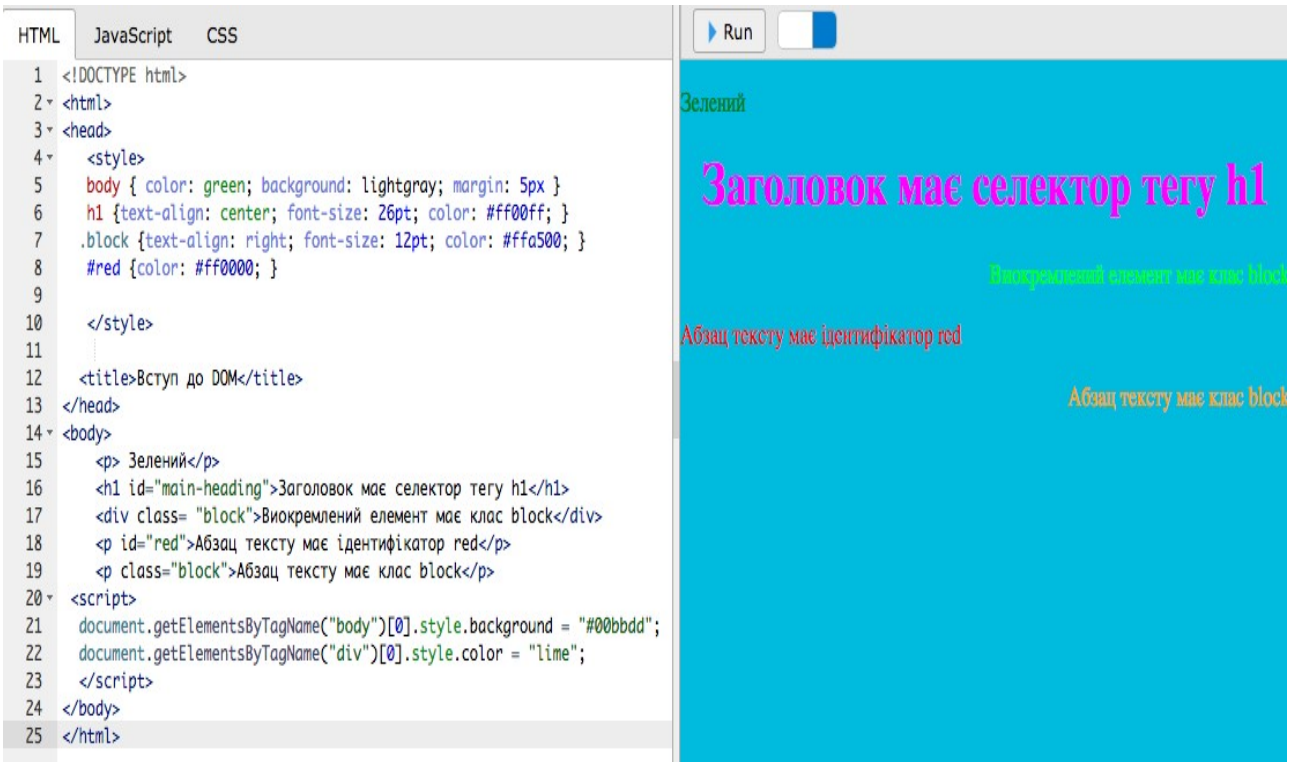


Рис. 6.25. Приклад виконання сценарію.

Керування відображенням вузла

Видимість чи невидимість вузла визначається через CSS, властивостями **`display`** або **`visibility`**.

Властивість ***visibility*** визначає, чи буде елемент приховано. Властивість призначена для відображення або приховування елемента, включаючи рамку навколо нього і фон. Під час приховування елементу, хоча він і стає прихованим, місце, яке він займає, залишається за ним.

Деякі значення властивості: **`visible / hidden`**

Приклад. Керування відображенням абзаців (див. Рис.6.26).

```
<!DOCTYPE html>
<html>
<head>
   <style>
      .vis {text-align: center; font-size: 12pt; color: #ffa500;
visibility: visible;}
      .hid {text-align: center; font-size: 12pt; color: #ffa500;
visibility: hidden;}
   </style>
</head>
<body>
<p class="vis">Це перший абзац. Він видимий</p>
<p class="hid">Це другий абзац. Він невидимий</p>
<p class="vis">Це третій абзац. Він теж видимий</p>
</body>
</html>
```

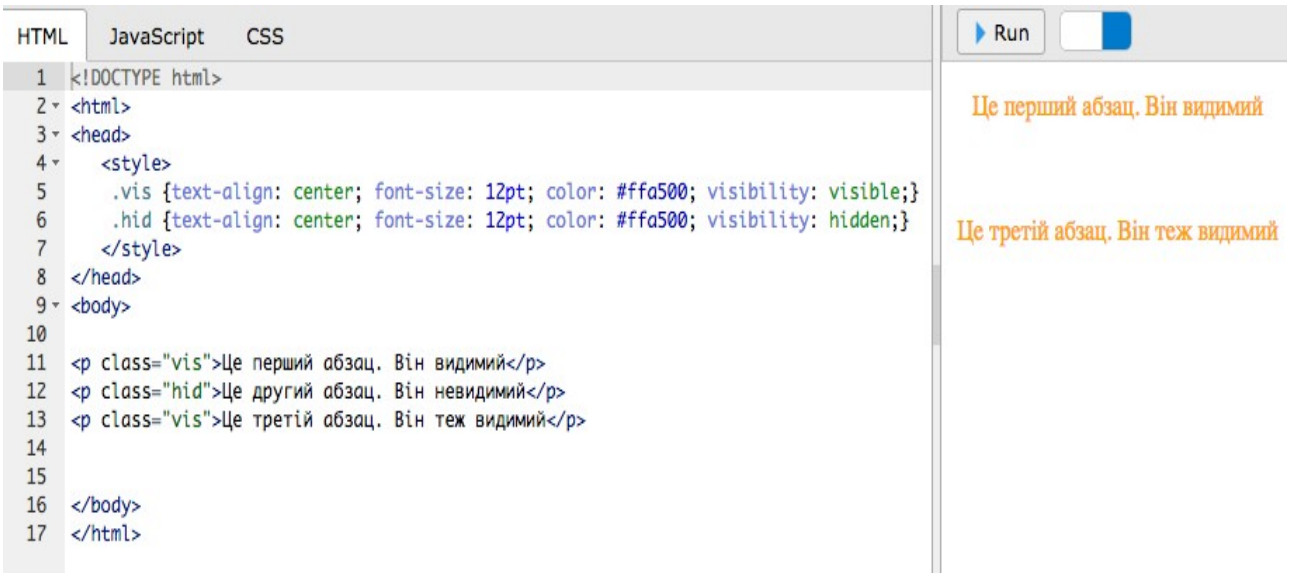


Рис. 6.26. Приклад виконання сценарію.

Властивість **`display`** вказує тип блока, який використовується для HTML-елемента.

Матеріали для самоосвіти:
https://css.in.ua/css/property/display

Серед значень властивості є **none**. Елемент приховується, ніби його не було. Зробити елемент видимим можна за допомогою скриптів. При цьому відбувається переформатування даних на сторінці з урахуванням знову доданого елемента.

Приклад. Маніпулювання властивістю display (див. Рис. 6.27).

```
<!DOCTYPE html>
<html>
<head>
   <style>
      .vis {text-align: center; font-size: 12pt; color: #ffa500;
visibility: visible;}
      .hid {text-align: center; font-size: 12pt; color: #ffa500;
display: none;}
   </style>
</head>
<body>
  <p class="vis">Це перший абзац. Він видимий</p>
  <p class="hid">Це другий абзац. Він невидимий</p>
  <p class="vis">Це третій абзац. Він теж видимий</p>
</body>
</html>
```

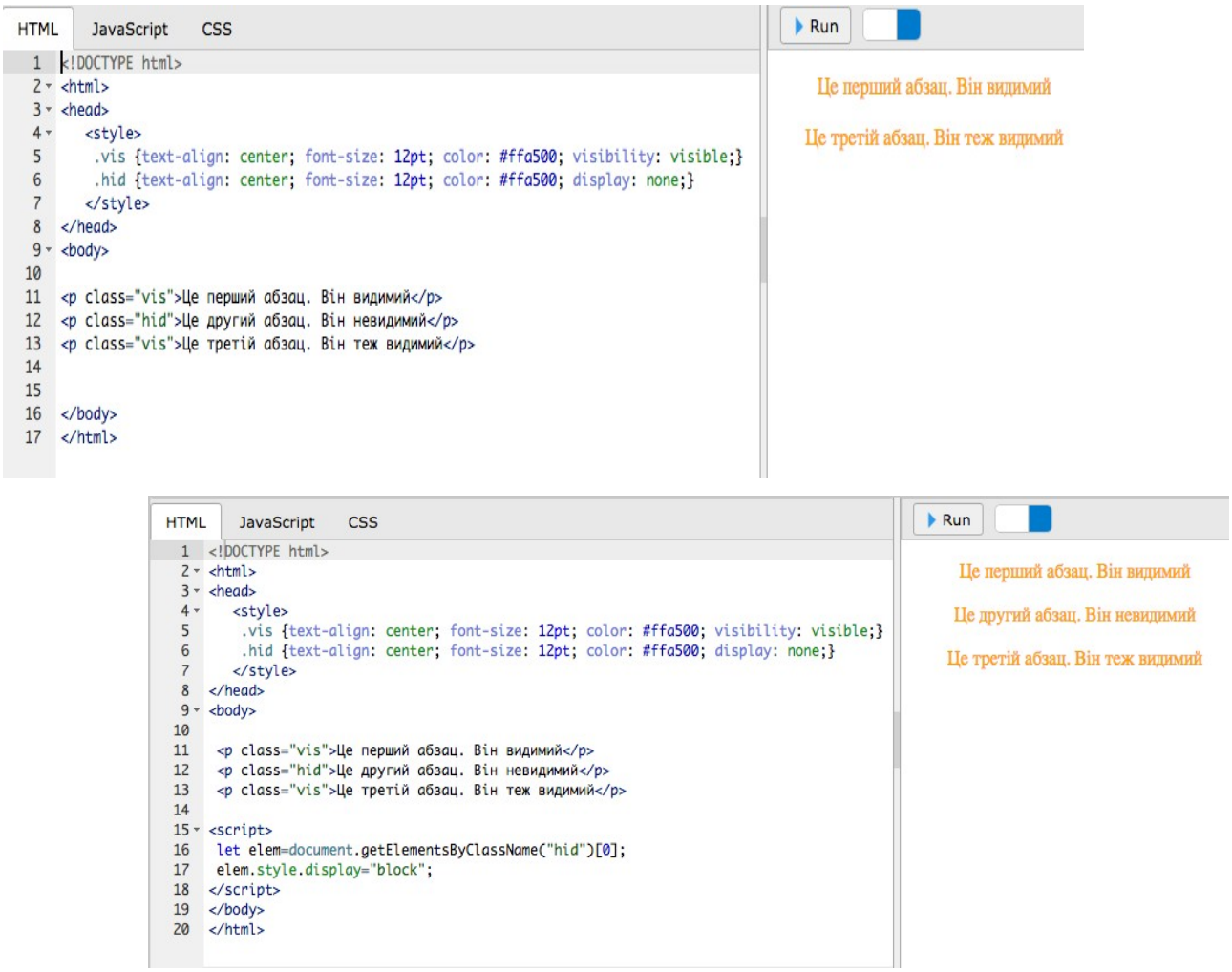


Рис. 6.27. Приклад виконання сценарію.

Властивість hidden

У стандарті HTML5 передбачено спеціальний атрибут і властивість для цього: hidden.

Приклад. Приховування абзаців за допомогою атрибута тегу і властивості елемента (див. Рис. 6.28).

```
<!DOCTYPE html>
<html>
<head>
   <style>
    .abc {text-align: center; font-size: 12pt; color: #ffa500; }
   </style>
</head>
<body>
<p class="abc">Це перший абзац.</p>
<p class="abc" hidden>Це другий абзац. У його тега є атрибут hidden</p>
<p class="abc">Це третій абзац. У нього буде властивість hidden</p>
<p class="abc">Це четвертий абзац.</p>
<p class="abc">Це п'ятий абзац.</p>
<p class="abc">Це шостий абзац.</p>
```

```
 <script>
   let parag3 = document.body.children[2];
   parag3.hidden = true;
 </script>
</body>
</html>
```

Рис. 6.28. Приклад виконання сценарію.

Внесення змін в HTML-документ

Матеріали для самоосвіти:
https://uk.javascript.info/modifying-document

Використовуючи DOM (об'єктну модель документа), можна створювати «живі» сторінки із змінним вмістом, додаючи нові елементи «на льоту» та змінюючи наявні.

Приклад. Виведення повідомлення на сторінку з використанням HTML (Рис. 6.29).

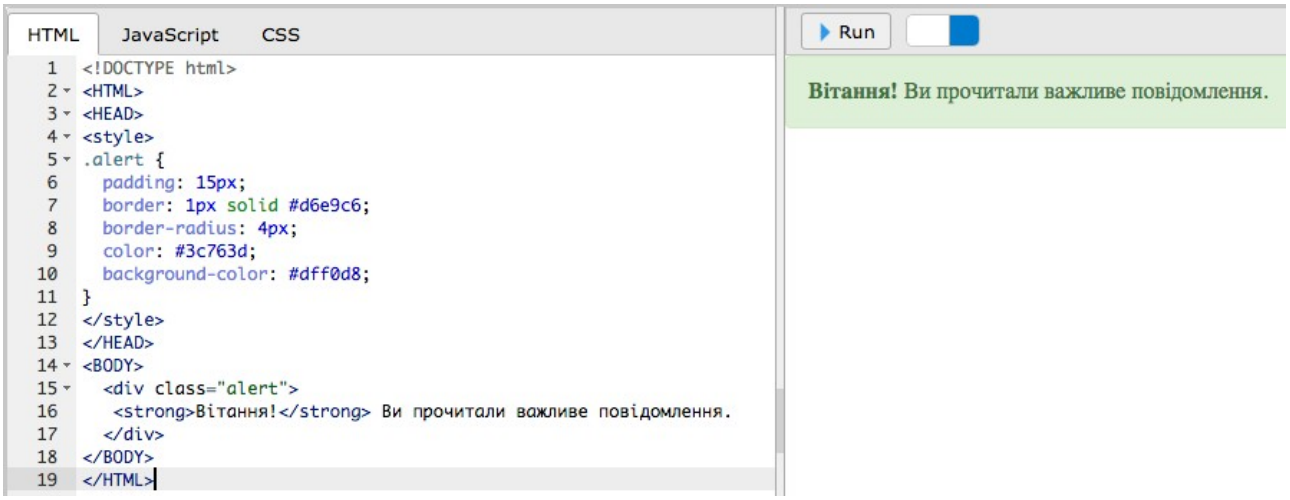


Рис. 6.29. Приклад виконання сценарію.

Такий самий `div` може бути створено за допомогою JavaScript (припускаємо, що стилі є в HTML або в окремому CSS-файлі).

Способи створення DOM вузлів:

`1.document.createElement(tag)`

Створює новий *елемент* з заданим тегом:

Приклад: `let div = document.createElement('div');`

`2. document.createTextNode(text)`

Створює новий *текстовий вузол* з заданим текстом:

Приклад: `let textNode = document.createTextNode('Новий текст');`

У більшості випадків створюються саме *елементи*, такі як `div` для повідомлень.

Створення повідомлень

Для створення елемента div для повідомлення за допомогою JavaScript слід виконати такі дії:

*1. Створити елемент *

```
let div = document.createElement('div');
```

2. Задати йому клас "alert"

```
div.className = "alert";
```

3. Наповнити змістом

```
div.innerHTML = "<strong>Вітання!</strong> Ви прочитали важливе повідомлення.";
```

Щоб елемент div можна було побачити, слід розмістити його на сторінці, наприклад, в елементі <body> (доступ до якого можна отримати за допомогою document.body), використавши метод append (див. Рис. 6.30):

`document.body.append(div)`

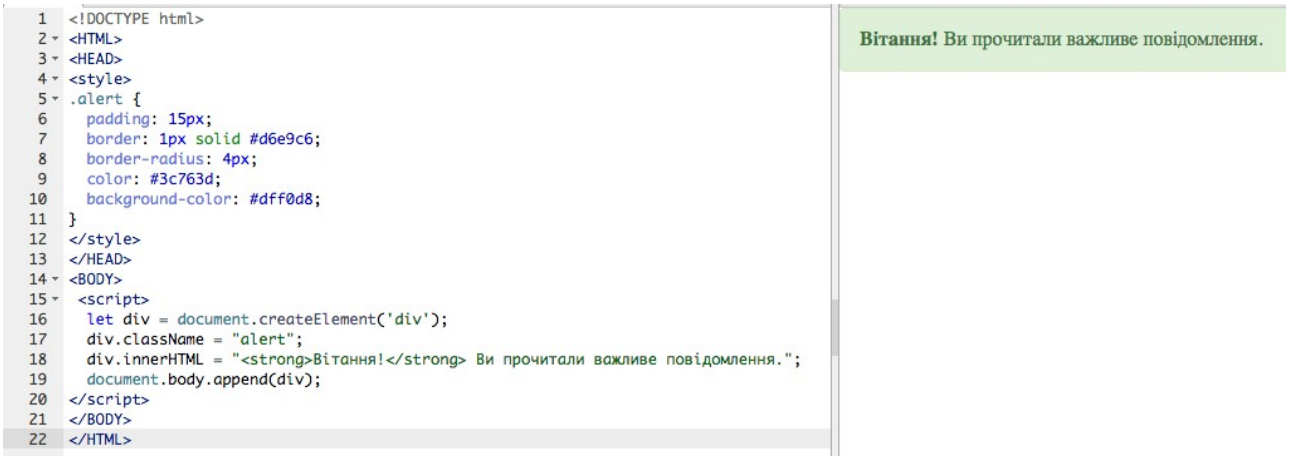


Рис. 6.30. Приклад виконання сценарію.

```
<!DOCTYPE html>
<HTML>
<HEAD>
<style>
.alert {
  padding: 15px;
  border: 1px solid #d6e9c6;
  border-radius: 4px;
  color: #3c763d;
  background-color: #dff0d8;
}
</style>
</HEAD>
<BODY>
 <script>
  let div = document.createElement('div');
  div.className = "alert";
  div.innerHTML = "<strong>Вітання!</strong> Ви прочитали важливе повідомлення.";
  document.body.append(div);
</script>
</BODY>
</HTML>
```

У розглянутому прикладі метод `append` був застосований до `document.body`, але можна застосувати його до будь-якого іншого елемента для вставки потрібного елемента всередину. Наприклад, за допомогою виклику `div.append(anotherElement)`, щось може бути додано до `<div>`.

Методи для вставки з указанням, куди саме буде вставлено вміст:

- **`node.append(...вузли або рядки)`** – додає вузли або рядки *в кінець* `node`,

- **`node.prepend(...вузли або рядки)`** – вставляє вузли або рядки *на початку* `node`,
- **`node.before(...вузли або рядки)`** – вставляє вузли або рядки *перед* `node`,
- **`node.after(...вузли або рядки)`** – вставляє вузли або рядки *після* `node`,
- **`node.replaceWith(...вузли або рядки)`** – замінює `node` заданими вузлами або рядками.

Аргументами цих методів може бути довільний список DOM вузлів або текстові рядки(які автоматично перетворюються на текстові вузли).

Приклад. Використання розглянутих методів для додавання до списку нових пунктів та тексту до/після нього (див. Рис. 6.31)

```
1  <!DOCTYPE html>
2  <HTML>
3  <HEAD>
4  </HEAD>
5  <BODY>
6  <ol id="ol">
7    <li>0</li>
8    <li>1</li>
9    <li>2</li>
10 </ol>
11  <script>
12   ol.before('before'); // вставити рядок "before" перед <ol>
13   ol.after('after'); // вставити рядок "after" після <ol>
14
15   let liFirst = document.createElement('li');
16   liFirst.innerHTML = 'prepend';
17   ol.prepend(liFirst); // вставити liFirst на початку <ol>
18
19   let liLast = document.createElement('li');
20   liLast.innerHTML = 'append';
21   ol.append(liLast); // вставити liLast в кінці <ol>
22 </script>
23 </BODY>
24 </HTML>
```

before

1. prepend
2. 0
3. 1
4. 2
5. append

after

Рис. 6.31. Приклад виконання сценарію.

```
<!DOCTYPE html>
<HTML>
<HEAD>
</HEAD>
<BODY>
<ol id="ol">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>
 <script>
  ol.before('before'); // вставити рядок "before" перед <ol>
  ol.after('after'); // вставити рядок "after" після <ol>
```

```
  let liFirst = document.createElement('li');
  liFirst.innerHTML = 'prepend';
  ol.prepend(liFirst); // вставити liFirst на початку <ol>

  let liLast = document.createElement('li');
  liLast.innerHTML = 'append';
  ol.append(liLast); // вставити liLast в кінці <ol>
</script>
</BODY>
</HTML>
```

Рис. 6.32 демонструє дію методів:

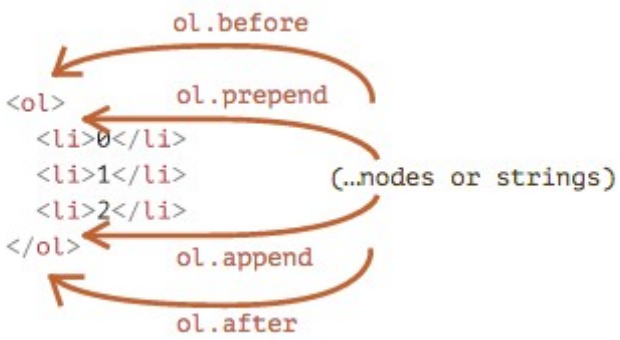


Рис. 6.32. Ілюстрація роботи методів

Остаточний вигляд списку:

```
before
<ol id="ol">
  <li>prepend</li>
  <li>0</li>
  <li>1</li>
  <li>2</li>
  <li>append</li>
</ol>
after
```

Вказані методи можуть вставляти декілька вузлів та фрагментів тексту за один виклик.

Приклад. Одночасна вставка рядка Hello та елемента hr (Рис. 6.33).

```
1  <!DOCTYPE html>
2  <HTML>
3      <HEAD>
4      </HEAD>
5      <BODY>
6      <div id="div1">block div1</div>
7      <script>
8       div1.before(document.createElement('hr'));
9       div1.after('Hello!',document.createElement('hr'))
10     </script>
11     </BODY>
12 </HTML>
```

block div1
Hello!

Рис. 6.33. Приклад виконання сценарію.

Якщо замість рядка Hello! вставити <p>Hello!<>/p>, теги у ньому сприйматимуться як звичайний текст, а не фрагмент html-документа (Рис. 6.34):

```
<!DOCTYPE html>
<HTML>
    <HEAD>
    </HEAD>
    <BODY>
    <div id="div1">block div1</div>
    <script>
     div1.before(document.createElement('hr'));
     div1.after('<p>Hello!</p>',document.createElement('hr'))
    </script>
    </BODY>
</HTML>
```

```
block div1
<p>Hello!</p>
```

Рис. 6.34. Приклад виконання сценарію.

Якщо потрібно вставити рядок HTML «як html», з усіма тегами та атрибутами (як це робить `elem.innerHTML`), використовується метод

`elem.insertAdjacentHTML(місце_вставки, html)`

Перший параметр (**`місце_вставки`**) - кодове слово, яке вказує, куди вставляти новий рядок відносно `elem`. Його значенням може бути:

- **`"beforebegin"`** – вставити `html` безпосередньо перед `elem`,
- **`"afterbegin"`** – вставити `html` в `elem`, на початку,
- **`"beforeend"`** – вставити `html` в `elem`, в кінці,
- **`"afterend"`** – вставити `html` безпосередньо після `elem`.

Другий параметр (**`html`**) **-** рядок у форматі HTML.

Приклад. Використання методу `insertAdjacentHTML (Рис. 6.35)`

```
<!DOCTYPE html>
<HTML>
    <HEAD>
    </HEAD>
    <BODY>
    <div id="div1">block div1</div>
    <script>
     div1.before(document.createElement('hr'));
     div1.after('<p>Hello!</p>',document.createElement('hr'));
     div1.insertAdjacentHTML('beforebegin', '<p>Hi!</p>');
     div1.insertAdjacentHTML('afterend', '<p>Bye!</p>');
    </script>
    </BODY>
</HTML>
```

```
Hi!

block div1

Bye!

<p>Hello!</p>
```

Рис. 6.35. Приклад виконання сценарію.

Строінка з кодом:

```
<div id="div1"></div>
<script>
  div1.insertAdjacentHTML('beforebegin', '<p>Hi!</p>');
  div1.insertAdjacentHTML('afterend', '<p>Bye!</p>');
</script>
```

виглядатиме так:

```
<p>Hi!</p>
<div id="div1"></div>
<p>Bye!</p>
```

Рис. 6.36 демонструє дію методу:

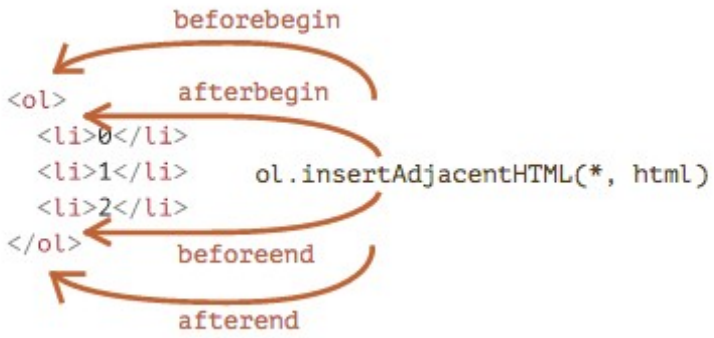


Рис. 6.36. Ілюстрація роботи методу.

Метод має двох братів:

- **`elem.insertAdjacentText(місце_вставки, текст)`** – синтаксис той самий, але рядок тексту вставляється «як текст» замість HTML;
- **`elem.insertAdjacentElement(місце_вставки, текст)`** – синтаксис той самий, але вставляється елемент.

Наведені методи існують для того щоб синтаксис залишався «однорідним». На практиці ж, у більшості випадків використовується **`insertAdjacentHTML`**, оскільки для вставки елементів та тексту є методи **`append/prepend/before/after`** – вони коротші для написання і так само вміють вставляти вузли чи фрагменти тексту.

Приклад. Виведення на екран повідомлення альтернативним способом (Рис. 6.37):

```
1  <!DOCTYPE html>
2  <HTML>
3      <HEAD>
4      <style>
5      .alert {
6       padding: 15px;
7       border: 1px solid #d6e9c6;
8       border-radius: 4px;
9       color: #3c763d;
10      background-color: #dff0d8;
11     }
12     </style>
13     </HEAD>
14 <BODY>
15 <script>
16   document.body.insertAdjacentHTML("afterbegin", `<div class="alert">
17     <strong>Всім привіт!</strong> Ви прочитали важливе повідомлення.
18   </div>`);
19 </script>
20 </BODY>
21 </HTML>
```

Всім привіт! Ви прочитали важливе повідомлення.

Рис. 6.37. Приклад виконання сценарію.

Видалення вузлів

Для видалення вузлів використовується метод **`node.remove()`**

Приклад. Повідомлення зникає через одну секунду (Рис. 6.38):

```
<!DOCTYPE html>
<HTML>
    <HEAD>
    <style>
    .alert {
     padding: 15px;
     border: 1px solid #d6e9c6;
     border-radius: 4px;
     color: #3c763d;
     background-color: #dff0d8;
    }
    </style>
    </HEAD>
<BODY>
<script>
  let div = document.createElement('div');
  div.className = "alert";
  div.innerHTML = "<strong>Всім привіт!</strong> Ви прочитали важливе повідомлення.";

  document.body.append(div);
  setTimeout(() => div.remove(), 1000);
</script>
</BODY>
</HTML>
```

Рис. 6.38. Приклад виконання сценарію.

```
<!DOCTYPE html>
<HTML>
<HEAD>
     <style>
    .alert {
     padding: 15px;
     border: 1px solid #d6e9c6;
     border-radius: 4px;
     color: #3c763d;
     background-color: #dff0d8;
    }
    </style>
</HEAD>
<BODY>
<script>
  let div = document.createElement('div');
  div.className = "alert";
    div.innerHTML = "<strong>Всім привіт!</strong> Ви прочитали
важливе повідомлення.";

  document.body.append(div);
  setTimeout(() => div.remove(), 1000);
</script>
</BODY>
</HTML>
```

Якщо потрібно *перемістити* елемент на інше місце в документі , немає потреби його видаляти.

Всі методи вставки автоматично видаляють вузол з попереднього місця розташування.

Приклад. Заміна елементів місцями (Рис. 6.39):

```
<!DOCTYPE html>
<HTML>
<HEAD>
</HEAD>
<BODY>
<div id="first">Перший</div>
<div id="second">Другий</div>
<script>
  // немає потреби викликати remove
  second.after(first); // взяти #другий та після нього вставити #перший
</script>
</BODY>
</HTML>
```

```
Другий
Перший
```

Рис. 6.39. Приклад виконання сценарію.

Клонування вузлів: cloneNode

Якщо в документі треба створити нове повідомлення, схоже на попереднє, можна використати такі способи:

1. Створити функцію та помістити код в неї.
2. К*лонувати* наявний `div` та змінити текст всередині (якщо потрібно).

У випадку, коли елемент великий, це може бути швидше і простіше.

Виклик **`elem.cloneNode(true)`** створює «глибоку» копію елемента – з усіма атрибутами та піделементами.

Виклик **`elem.cloneNode(false)`** створює копію без дочірніх елементів.

Приклад. Копіювання повідомлення (Рис. 6.40).

```
<!DOCTYPE html>
<HTML>
<HEAD>
<style>
.alert {
  padding: 15px;
  border: 1px solid #d6e9c6;
  border-radius: 4px;
  color: #3c763d;
  background-color: #dff0d8;
}
</style>
</HEAD>
<BODY>
<div class="alert" id="div">
  <strong>Всім привіт!</strong> Ви прочитали важливе повідомлення.
</div>

<script>
  let div2 = div.cloneNode(true); // клонувати елемент
  div2.querySelector('strong').innerHTML = 'До побачення!'; // змінити клона

  div.after(div2); // вставити клонований елемент після наявного `div`
</script>
</BODY>
</HTML>
```

Всім привіт! Ви прочитали важливе повідомлення.

До побачення! Ви прочитали важливе повідомлення.

Рис. 6.40. Приклад виконання сценарію.

Застарілі методи вставки/видалення

1. `parentElem.appendChild(node)`

Додає `node` як останній дочірній елемент `parentElem`.

Приклад. Додавання `<li>` в кінець `<ol>` (Рис.6.41):

```
<!DOCTYPE html>
<HTML>
<HEAD>
</HEAD>
<BODY>
<ol id="list">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>

<script>
  let newLi = document.createElement('li');
  newLi.innerHTML = 'Привіт, світ!';

  list.appendChild(newLi);
</script>
</BODY>
</HTML>
```

1. 0
2. 1
3. 2
4. Привіт, світ!

Рис. 6.41. Приклад виконання сценарію.

2. `parentElem.insertBefore(node, nextSibling)`

Вставляє `node` перед `nextSibling` в `parentElem`.

Приклад. Вставка нового елемента списку перед другим `<li>` (Рис.6.42):

```
<!DOCTYPE html>
<HTML>
<HEAD>
</HEAD>
<BODY>
<ol id="list">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>
<script>
  let newLi = document.createElement('li');
  newLi.innerHTML = 'Привіт, світ!';

  list.insertBefore(newLi, list.children[1]);
</script>
</BODY>
</HTML>
```

1. 0
2. Привіт, світ!
3. 1
4. 2

Рис. 6.42. Приклад виконання сценарію.

Вставити `newLi` першим елементом можна так:

```
list.insertBefore(newLi, list.firstChild);
```

3. `parentElem.replaceChild(node, oldChild)`

Замінює `oldChild` на `node` поміж дочірніми елементами `parentElem`.

4. `parentElem.removeChild(node)`

Видаляє `node` з `parentElem` (припускаючи, що `node` – це його дочірній елемент).

Приклад. Видалення першого `<li>` з `<ol>` (Рис.6.43)::

```
<!DOCTYPE html>
<HTML>
<HEAD>
</HEAD>
<BODY>
<ol id="list">
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>

<script>
  let li = list.firstElementChild;
  list.removeChild(li);
</script>
</BODY>
</HTML>
```

```
1. 1
2. 2
```

Рис. 6.43. Приклад виконання сценарію.

Всі ці методи повертають вставлені/видалені вузли. Наприклад, **`parentElem.appendChild(node)`** повертає `node`. Але зазвичай повернуті значення не використовуються.

7. ВСТУП ДО ПОДІЙНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Подійно-орієнтоване програмування (event-driven programming) - підхід до програмування, в якому виконання програми визначається подіями - діями користувача (наприклад, з клавіатурою, мишею, джойстиком тощо), повідомленнями інших програм і потоків, подіями операційної системи (наприклад, надходженням мережевого пакета).

ПОП можна також визначити як спосіб побудови комп'ютерної програми, при якому код описує отримання повідомлення про подію і опрацювання події.
Події та опрацювання подій є важливою частиною для програмування мовою JavaScript.
Сценарії в документі HTML призначені, зокрема, для ***обробки подій.***

Стандартні події браузера
Події миші

'mousedown' – подія mousedown викликається, коли вказівний пристрій (зазвичай мишка) натискається вниз над елементом.

'mouseup' – подія mouseup викликається, коли вказівний пристрій (зазвичай мишка) відпускається вгору над елементом.

'click' – подія click визначається як mousedown, за яким слідує mouseup в тому ж положенні.

'dblclick' – ця подія викликається, коли на елементі клацають двічі у швидкій послідовності в одній і тій же позиції (подвійний клік).

'mouseover' – подія mouseover викликається, коли пристрій переміщається над елементом.

'mouseout' – подія mouseout викликається, коли вказівний пристрій переміщається в сторону від елемента.

'mousemove' – подія mousemove викликається, коли вказівний пристрій переміщається при наведенні на елемент.

Події клавіатури

'keypress' – ця подія викликається при кожному натисканні клавіші на клавіатурі.

'keydown' – ця подія також спрацьовує при кожному натисканні клавіші, вона запускається до події keypress.

'keyup' – ця подія викликається при відпусканні клавіші після подій keydown і keypress.

Події форми

'select' – ця подія викликається, коли обирається текст всередині текстового поля (input, textarea і т.д.).

'change' – ця подія викликається, коли елемент управління втрачає фокус введення і/або значення було змінено з моменту отримання фокусу.

'submit' – ця подія викликається при відправленні форми.

'reset' – ця подія викликається при скиданні форми.

'focus' – ця подія викликається, коли елемент отримує фокус (зазвичай від курсора миші).

'blur' – ця подія викликається, коли елемент втрачає фокус.

Інші події

'load' – ця подія викликається, коли браузер завершив завантаження всього вмісту в документі, включаючи контент, картинки, фрейми і об'єкти.

'resize' – ця подія викликається при зміні розміру документа (тобто коли змінюється розмір браузера).

'scroll' – ця подія викликається при прокручуванні документа.

'unload' – ця подія викликається, коли браузер видаляє весь контент з вікна або фрейму, тобто коли користувач покидає сторінку.

Визначення обробника події

Більшість тегів HTML мають спеціальні *атрибути, що визначають події*, на які можуть реагувати відповідні елементи.

Такий атрибут починається з **"on"**, а далі вказується назва події:
"on<Подія>": onLoad, onClick.

Значенням такого атрибуту-події є рядок, що містить *сценарій - код-обробник події.*

На практиці JavaScript-код часто буває занадто об'ємним для прямого розміщення у тегу. Тоді код доцільно оформити у функцію і викликати *функцію як обробник події.*

****

Приклад. Опрацювання події Click (клацання кнопкою) відповідною програмою onClick для створення вікна з повідомленням:

```
<form>
<input type="button"
value="Натисніть на кнопку"
onClick="alert('Обробник події Click')">
</form>
```

Тут засобами HTML створено форму з кнопкою.

Атрибут тега <input> - *onClick="alert('Обробник події Click')"* - визначає відповідь на подію натискання на кнопку з написом "Натисніть на кнопку".

Такою відповіддю є виклик alert('Обробник події Click') (див. Рис. 7.1)

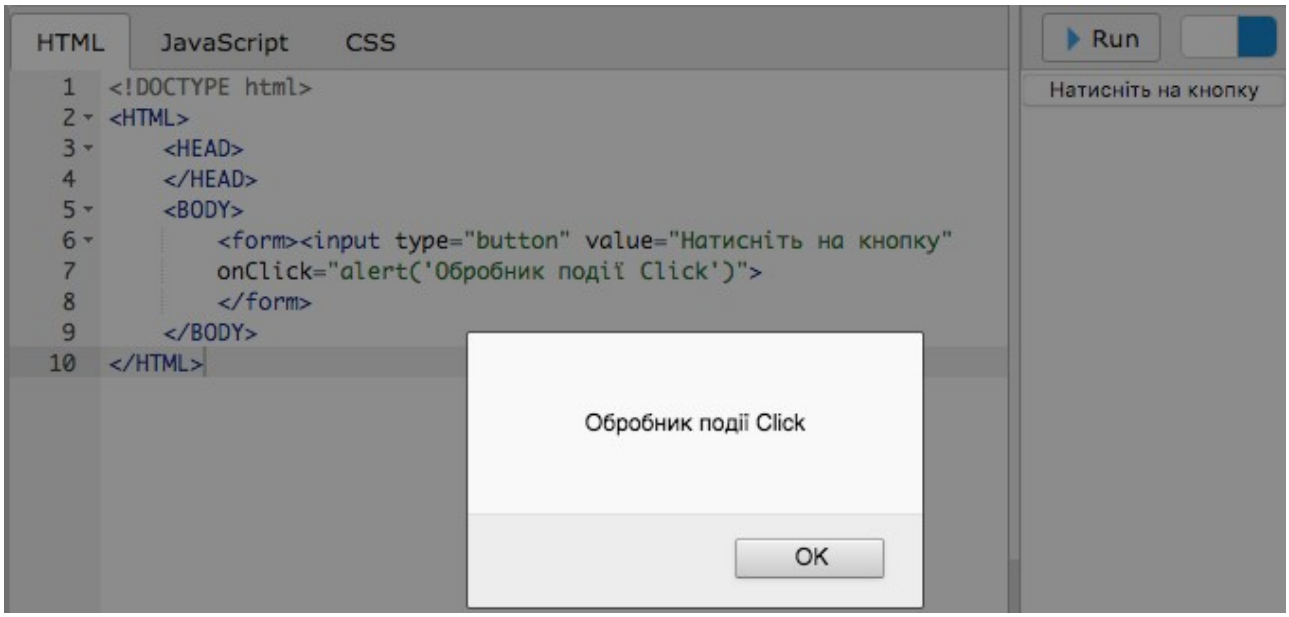


Рис. 7.1. Приклад виконання сценарію.

Планування виклику функцій
Матеріали для самоосвіти
https://uk.javascript.info/settimeout-setinterval

У попередніх прикладах JavaScript починав роботу відразу після завантаження сторінки, зупиняючись лише при виклику деяких функцій. Однак інколи не потрібно виконувати весь код відразу, а треба запустити фрагмент коду через якийсь час або у відповідь на дію користувача.

Розглянемо способи керувати тим, коли виконується код. Це дає змогу створювати інтерактивні вебсторінки, які можуть змінюватися з часом і реагувати на дії користувачів.

Відкладене виконання коду та setTimeout

Замість виклику функції відразу, можна вказати JavaScript зробити це через певний час. Таке відкладене виконання називається запуском за таймером. Для цього в JavaScript є функція setTimeout.

`setTimeout(func, timeout)`

Аргументи:

func - функція, яку треба буде викликати при спрацьовуванні таймера (через timeout мілісекунд),

timeout - час очікування в мілісекундах (перед запуском функції).

Приклад. Виведення вікна alert через setTimeout (див. Рис. 7.2).

```
var timeUp = function () {
    alert("Запуск функції через 2 секунди!");
};
setTimeout(timeUp, 2000);
```

Виклик setTimeout повертає число - ідентифікатор (ID) таймера, яке змінюється при наступному виклику функції. На рисунку 7.2. показано, як це

число виводиться в консоль розробника у браузері. Це число можна використати для скасування цього конкретного таймера.

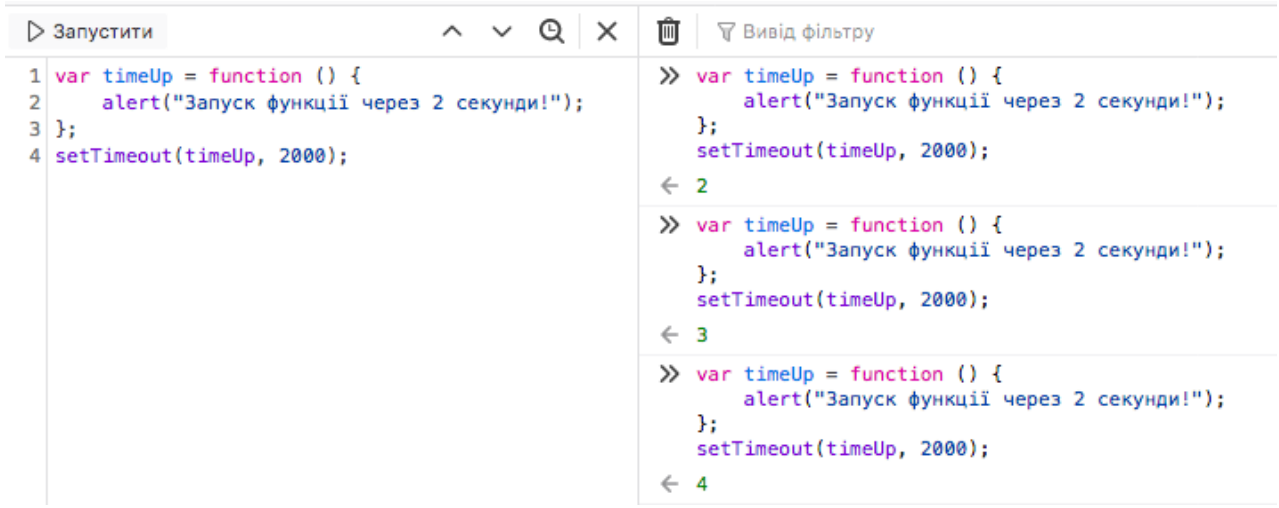


Рис. 7.2. Приклад виконання сценарію.

Скасування дії таймера

Після задання відкладеного виклику функції за допомогою setTimeout може з'ясуватися, що викликати цю функцію не потрібно.

Приклад. Встановлено нагадування про виконання певного завдання. Але завдання виконано завчасно, і нагадування вже не потрібне.

Для скасування дії таймера використовується функція **`clearTimeout`**, аргументом якої є ID таймера, отриманий раніше від setTimeout.

Нагадування встановлено таким чином:

```
var reminder = function () {
      alert("Не забудьте про це завдання!");
      };
var timeoutId = setTimeout(reminder, 60000);
```

Функція reminder створює діалог alert, що нагадує про завдання. Виклик setTimeout(reminder, 60000) повідомляє JavaScript, що функцію reminder потрібно викликати через 60 000 мілісекунд (тобто 60 секунд). В останньому рядку викликається setTimeout і ID таймера зберігається в новій змінній timeoutID.

Для скасування дію таймера слід передати його ID функції clearTimeout:

```
clearTimeout(timeoutId);
```

Тепер setTimeout не буде викликати функцію reminder (див. Рис. 7.3).

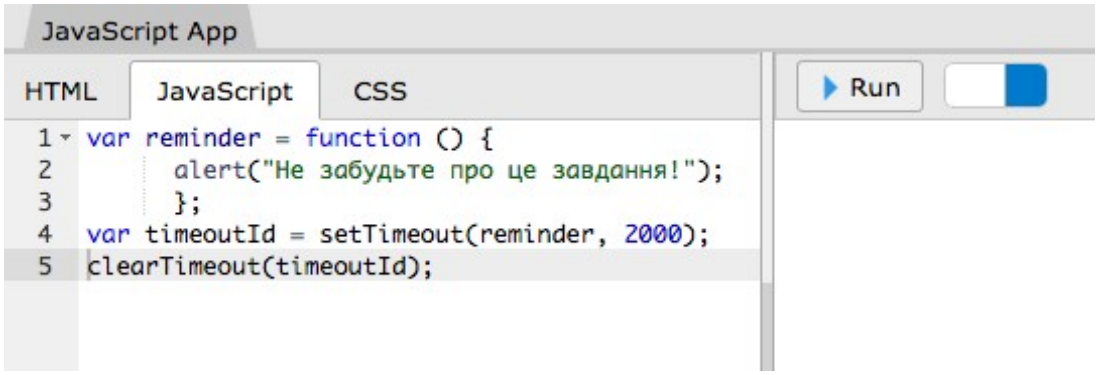


Рис. 7.3. Приклад виконання сценарію.

Багаторазовий запуск коду та setInterval

Функція setInterval схожа на setTimeout, але вона викликає передану їй функцію повторно через певні проміжки (інтервали) часу.

Приклад. Щоб оновлювати покази годинника за допомогою JavaScript, можна використати setInterval, щоб функція оновлення викликалася раз на секунду.

`setInterval(func, interval)`

Аргументи:

func - функція, яку треба викликати кожні interval мілісекунд,

interval - час між викликами функції func в мілісекундах.

Так само як setTimeout повертає ID таймера, setInterval повертає ID інтервалу. Його можна використовувати для скасування періодичного виклику функції за допомогою функції **`clearInterval.`**

Приклад. Написати код, за яким в консоль браузера виводиться певний текст з інтервалом у 2 секунди. З 5 секунд припинити виведення.

```
// почати виведення тексту з інтервалом 2 секунди
function work() {
    console.log('work');
}
var timerID=setInterval(work,2000);

//за 5 секунд зупинити виведення тексту

function stop(){
    clearInterval(timerID);
    console.log('stop');
}
setTimeout(stop,5000);
```

Приклад. Щосекундне виведення повідомлення в консоль із зупинкою через заданий час (див. Рис. 7.4).

```
let counter=1;
const STOP=10;
let printMessage = function() {
 console.log("Ви дивитесь в консоль вже " + counter + " сек");
 counter++;
  };
let intervalId = setInterval(printMessage, 1000);

function stopPrint(){
 clearInterval(intervalId);
 }

setTimeout(stopPrint,STOP*1000);
```

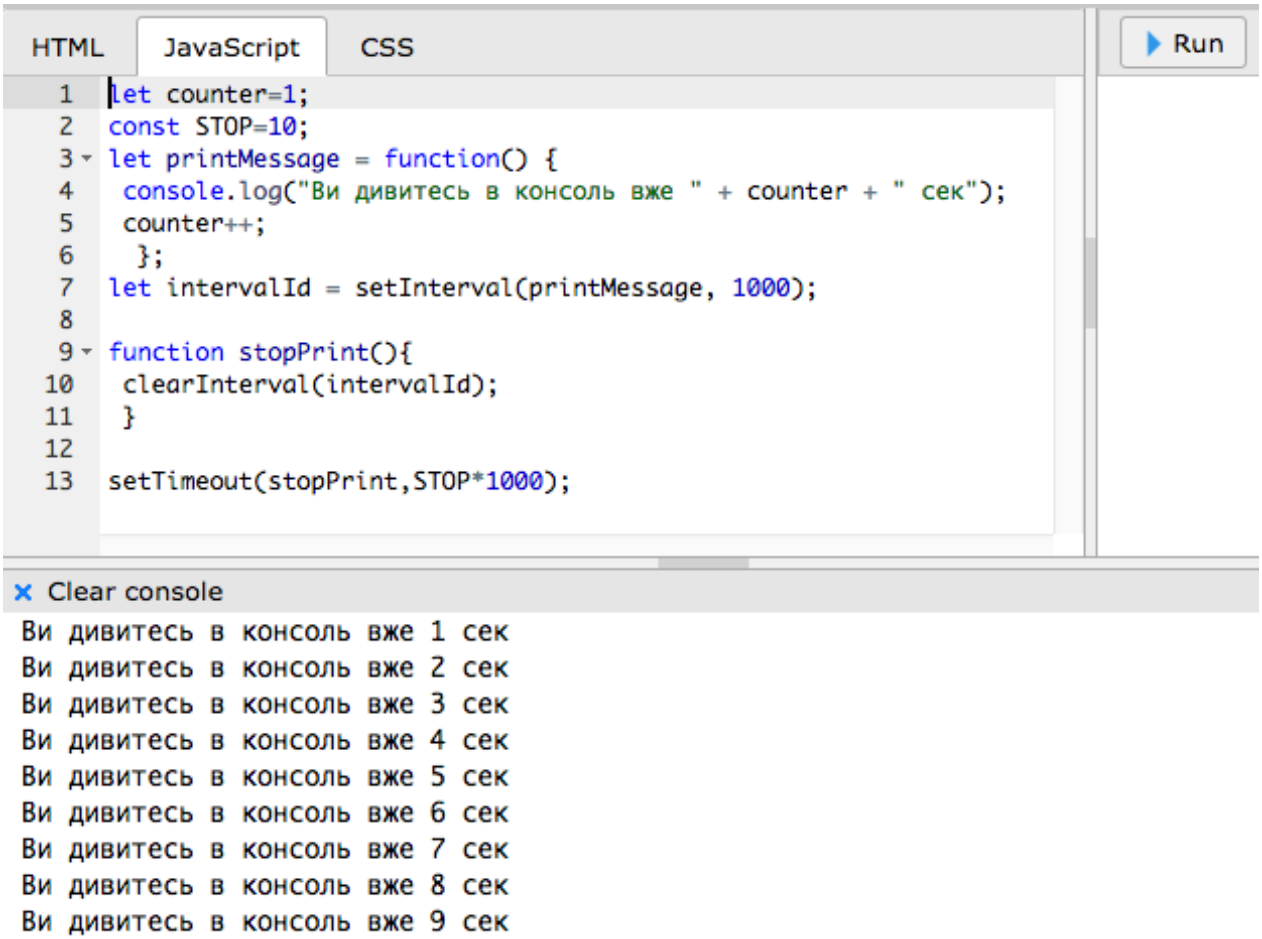


Рис. 7.4. Приклад виконання сценарію.

Приклад. Нехай є документ такої структури (див. Рис. 7.5)

```
<HTML>
<HEAD>
 <style>
        #p1 {text-align: left; background-color: Yellow;}
        #p2 {text-align: center; background-color: LightBlue;}
        #p3 {text-align: right; background-color: Pink;}
        .d1 {text-align: left; font-size: 12pt; color: #FF0000;}
        .d2 {text-align: left; font-size: 14pt; color: #00FF00;}
        .d3 {text-align: left; font-size: 16pt; color: #0000FF;}
        .d4 {text-align: left; font-size: 18pt; color: #777777;}
      </style>
</HEAD>
<BODY>
<p id="p1">Цей абзац має ідентифікатор p1</p>
<p id="p2">Цей абзац має ідентифікатор p2</p>
<p id="p3">Цей абзац має ідентифікатор p3</p>
   <div class="d1">Блок 1</div>
   <div class="d2">Блок 2</div>
   <div class="d3">Блок 3</div>
   <div class="d4">Блок 4</div>
</BODY>
</HTML>
```

Рис. 7.5. Приклад документа.

Завдання 1. Написати код, за яким при натисненні лівої кнопки миші на абзаці з ідентифікатором p2 буде виводитись вікно з текстом цього абзацу.

У код на Рис.7.2. слід внести такі доповнення:

```
<script>
    function info1(){
       let elem=document.getElementById('p2');
       alert(elem.innerHTML);
    }
</script>
...
<p id="p1">Цей абзац має ідентифікатор p1</p>
<p id="p2" onClick="info1()">Цей абзац має ідентифікатор p2</p>
<p id="p3">Цей абзац має ідентифікатор p3</p>
```

Приклад виконання Завдання 1 наведено на Рис. 7.6.

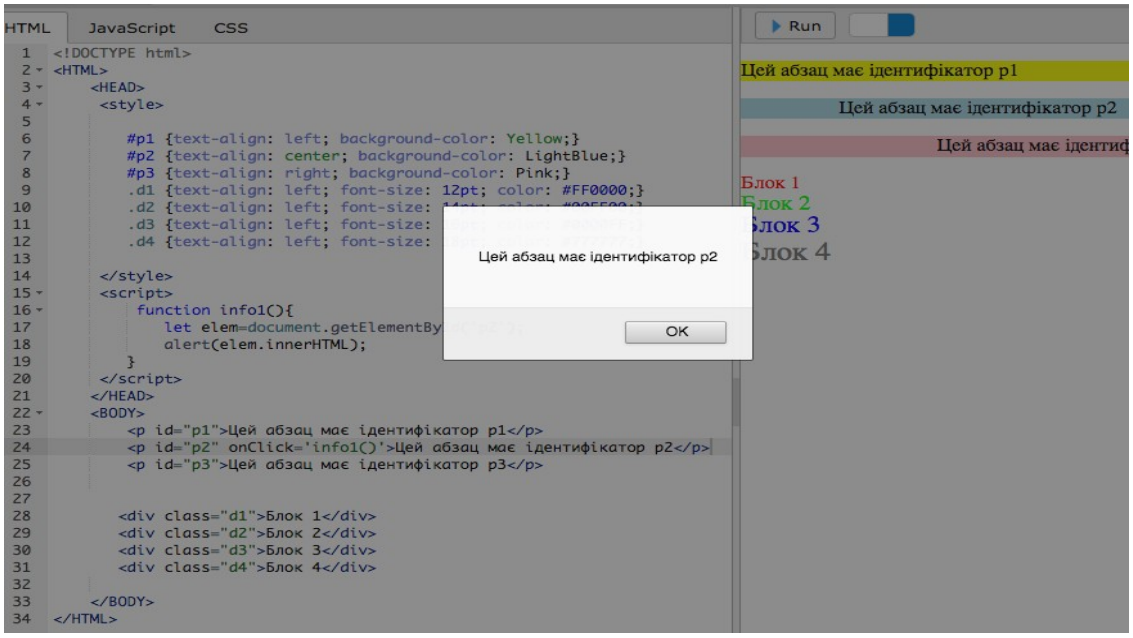


Рис. 7.6. Приклад виконання Завдання 1.

Завдання 2. Написати код, за яким при натисненні лівої кнопки миші на останньому з абзаців колір тексту першого абзацу буде змінено на червоний.

У код на Рис.7.5. слід внести такі доповнення:

```
<script>
   function info2(){
    let elem=document.getElementsByTagName('p');
    elem[0].style.color="Red";
     }
</script>
...
<p id="p1">Цей абзац має ідентифікатор p1</p>
<p id="p2" onClick="info1()">Цей абзац має ідентифікатор p2</p>
<p id="p3" onClick="info2()">Цей абзац має ідентифікатор p3</p>
```

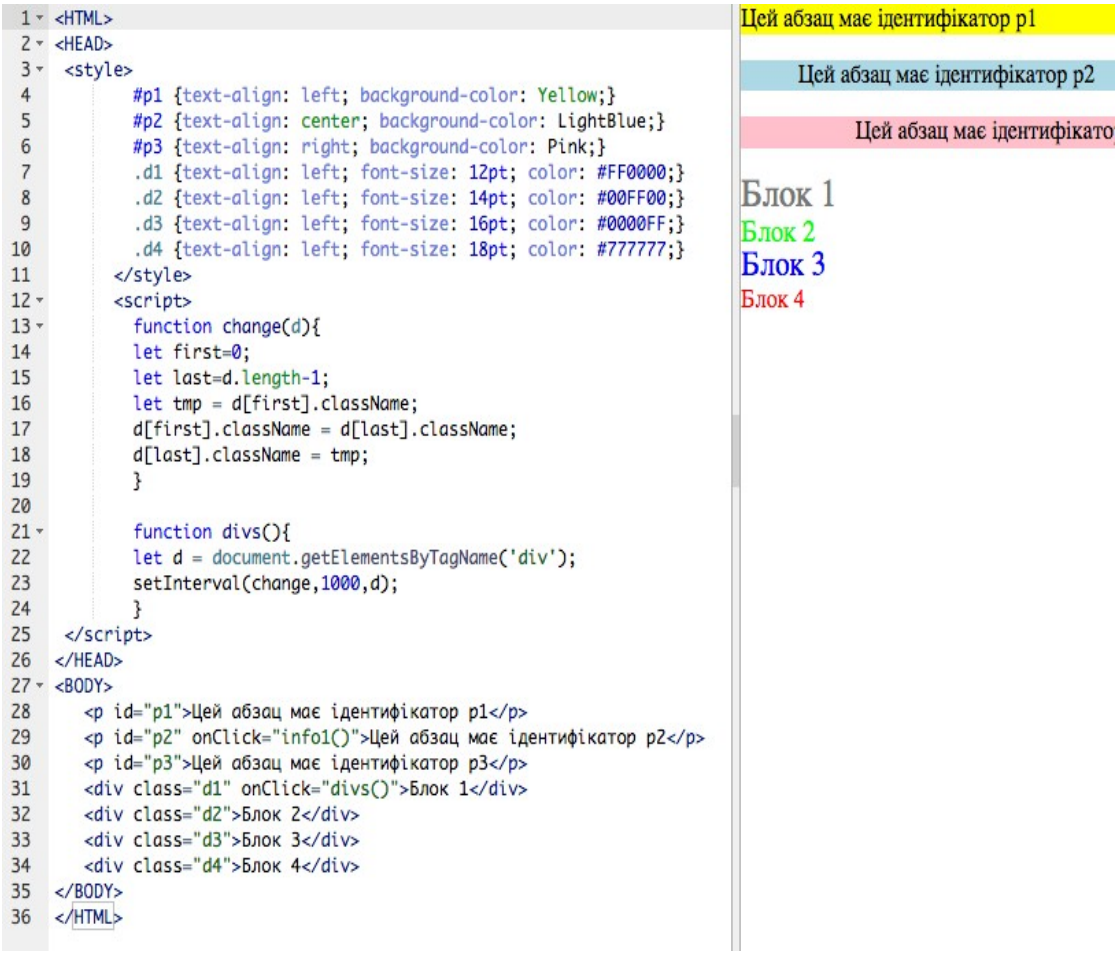
Завдання 3. Написати код, за яким при натисненні лівої кнопки миші на першому з блоків форматування першого і останнього блоків будуть мінятись місцями з інтервалом 1 секунду.

У код на Рис.7.5. слід внести такі доповнення:

```
<script>
        function change(d){
        let first=0;
        let last=d.length-1;
        let tmp = d[first].className;
        d[first].className = d[last].className;
        d[last].className = tmp;
        }

        function divs(){
        let d = document.getElementsByTagName('div');
        setInterval(change,1000,d);
       }
 </script>
...
 <div class="d1" onClick="divs()">Блок 1</div>
```

Приклад виконання Завдання 3 наведено на Рис. 7.7.




```
<HTML>
<HEAD>
 <style>
        #p1 {text-align: left; background-color: Yellow;}
        #p2 {text-align: center; background-color: LightBlue;}
        #p3 {text-align: right; background-color: Pink;}
        .d1 {text-align: left; font-size: 12pt; color: #FF0000;}
        .d2 {text-align: left; font-size: 14pt; color: #00FF00;}
        .d3 {text-align: left; font-size: 16pt; color: #0000FF;}
        .d4 {text-align: left; font-size: 18pt; color: #777777;}
      </style>
      <script>
        function change(d){
        let first=0;
        let last=d.length-1;
        let tmp = d[first].className;
        d[first].className = d[last].className;
        d[last].className = tmp;
        }

        function divs(){
        let d = document.getElementsByTagName('div');
        setInterval(change,1000,d);
        }
 </script>
</HEAD>
<BODY>
   <p id="p1">Цей абзац має ідентифікатор p1</p>
   <p id="p2" onClick="info1()">Цей абзац має ідентифікатор p2</p>
   <p id="p3">Цей абзац має ідентифікатор p3</p>
   <div class="d1" onClick="divs()">Блок 1</div>
   <div class="d2">Блок 2</div>
   <div class="d3">Блок 3</div>
   <div class="d4">Блок 4</div>
</BODY>
</HTML>
```

Рис. 7.7. Приклад виконання Завдання 3.

8. ФОРМИ

Вебформи призначені для обміну даними між користувачем та сервером.

Форма є розділом документа, що містить інтерактивні елементи управління, які дозволяють користувачеві відправляти дані на вебсервер.

Область застосування форм не обмежена відправленням даних на сервер - за допомогою клієнтських скриптів можна отримати доступ до будь-якого елемента форми, змінювати його та застосовувати на власний розсуд.

Для вказання браузеру, де починається і закінчується форма, використовується контейнер **<FORM>**. Між тегами <FORM> та </FORM> можна розміщувати будь-які необхідні теги HTML. Це дозволяє додати елементи форми до комірок таблиці для їх форматування, а також використовувати зображення. Документ може містити кілька форм, але вони не повинні бути вкладені одна в одну.

Матеріали для самоосвіти:
https://css.in.ua/html/tag/form
https://w3schoolsua.github.io/html/html_forms.html

Параметри форми:

- Стандартні поля для введення інформації.
- Кнопка надсилання даних форми на сервер (кнопка SUBMIT).
- Адреса програми на вебсервері, яка оброблятиме вміст даних форми.

Коли форма надсилається на сервер, керування даними передається програмі, заданій атрибутом action елемента <form>.

Попередньо браузер готує дані у вигляді пари *"ім'я=значення"*, де ім'я визначається атрибутом name елемента <input>, а значення введено користувачем або встановлено в полі за замовчуванням.

Таблиця 8.1.

Деякі атрибути форм

Атрибут	Значення	Опис
method	Визначає HTTP метод надсилання даних форми (за замовчуванням використовується метод GET)	
	get	Передає дані форми в адресному рядку («ім'я=значення»), які додаються до URL сторінки після знаку запитання і розділяються між собою амперсандом (&). (http://example.ua/doc/?name=Ivan&password=vanya)
	post	Передає дані форми на сервер для обробки у тілі запиту http.
name	text	Визначає ім'я форми.
novalidate	novalidate	Встановлює, що дані, введені у форму, не будуть перевірятися перед відправленням.

Продовження Таблиці 8.1.

Атрибут	Значення	Опис
target	Визначає, де показати відповідь, отриману після відправлення форми	
	_blank	показує в новому вікні.
	_self	показує в поточному вікні.
	_parent	показує в батьківському вікні.
	_top	відкриває на всю ширину вікна.
	framename	відкриває у фреймі (ім'я має бути вказано як значення)
accept	file_type	Визначає розділений комами список типів файлів, які можуть бути завантажені на сервер. Не підтримується в HTML5.
accept-charset	character_set	Вказує кодування даних, які відправляються на сервер.
action	URL	Вказує URL адресу обробника форми.
autocomplete	on off	Вмикає/вимикає автозаповнення полів форми. За замовчуванням значення увімкнено.
enctype	application/x-www-form-urlencoded multipart/form-data text/plain	Визначає спосіб кодування даних форми при їх надсиланні. (За замовчуванням application/x-www-form-urlencoded). (Використовується лише за методом POST).

Атрибут method вказує браузеру, який вид HTTP запиту необхідно використовувати для надсилання форми; можливі значення POST та GET.

Метод **GET** є одним з найбільш поширених і призначений для отримання необхідної інформації і передачі даних в адресному рядку. Пари «ім'я=значення» приєднуються в цьому випадку до адреси після знаку питання і розділяються між собою амперсандом (символ &).

Зручність використання методу отримання полягає в тому, що адресу з усіма параметрами можна використовувати неодноразово, зберігши його, наприклад, в закладки браузера, а також змінювати значення параметрів прямо в адресному рядку.

Довжина URL обмежена (близько 3000 знаків, 4kb);
Цей метод не використовують для передавання конфіденційних даних(їх буде видно в URL).

Метод **POST** призначений для відправлення інформації на сервер.

Дописує дані форми в тіло HTTP запиту (дані не видно в URL).

Не має обмежень за розміром.

Великі обсяги даних використовуються в форумах, поштових службах, при заповненні бази даних, при пересиланні файлів та ін.

Атрибут name визначає унікальне ім'я форми.

Як правило, назва форми використовується для доступу до її елементів через скрипти. Як ім'я використовується набір символів, включаючи числа та літери. Java Script чутливий до регістру, тому при зверненні до форми за ім'ям через скрипти слід використовувати таке саме написання, що і в параметрі name.

Атрибут enctype встановлює тип даних, що відправляються разом з формою.

Зазвичай встановлювати значення параметра enctype не потрібно, дані правильно розуміються на стороні сервера. Однак, якщо використовується поле для надсилання файлу (<INPUT type="file">), слід визначити параметр enctype як multipart/form-data (див. приклад). Допускається також встановлювати відразу кілька значень, розділяючи їх комами.

Приклад форми (див. Рис.8.1)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
  </head>
  <body>
   <form name="form1" method="post" action="../admin/add.php">
   Тут розміщені елементи форми
   <input type="submit">
   </form>
       <form name="form2">
       Тут розміщені елементи форми
       <input type="submit">
       </form>
<form          name="form3"          action="/cgi-bin/handler.cgi"
enctype="multipart/form-data" method="POST">
Тут розміщені елементи форми
</form>
</body>
</html>
```

Рис.8.1. Приклад форми.

Після того, як обробник форми отримує дані, він повертає результат як HTML-документ. Можна визначити вікно, в яке завантажуватиметься підсумкова вебсторінка. Для цього використовується параметр **target**, його значенням є ім'я вікна або кадру. Якщо параметр target не встановлений, результат, що повертається, показується в поточному вікні. В якості аргументу використовується ім'я вікна або кадру, задане параметром name. Якщо встановлено ім'я, що не існує, відкриється нове вікно.

Зарезервовані імена:

_blank - завантажує сторінку у нове вікно браузера.

_self – завантажує сторінку у поточне вікно.

_parent - завантажує сторінку у фрейм-батько, якщо фреймів немає, цей параметр працює як _self.

_top - скасовує всі фрейми і завантажує сторінку у повному вікні браузера, якщо фреймів немає, цей параметр працює як _self.

Приклад.

```
<form name=”form3” target="_blank">
```

Тег <input>

Тег <input> є одним з елементів форми і дозволяє створювати різні елементи інтерфейсу та забезпечувати взаємодію з користувачем. Переважно <input> призначений для створення текстових полів, різних кнопок, перемикачів та прапорців.

Хоча елемент <input> не потрібно розміщувати всередині контейнера <FORM>, що визначає форму, але якщо введені користувачем дані повинні бути відправлені на сервер, де їх обробляє серверна програма, то вказувати <FORM> обов'язково. Те саме відбувається і в разі обробки даних за допомогою клієнтських додатків, наприклад, скриптів JavaScript.

Тег <input> використовується в межах елемента <FORM>, визначаючи поля для введення інформації користувачем. Тип поля (текст, радіокнопка, прапорець, поле введення пароля тощо) визначається значенням атрибута type.

Тег <input> непарний. Тег не містить текстовий контент, лише атрибути.

Приклад. Фрагмент документа з використання тегу <input> (див. Рис. 8.2)

```
<form action="getform.php" method="get">
 Ім"я: <input type="text" name="first_name">
 Прізвище: <input type="text" name="last_name">
 E-Mail: <input type="email" name="user_email">
 <input type="submit" value="Відправити">
</form>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок документа</title>
  </head>
  <body>
    <form action="getform.php" method="get">
      Ім"я: <input type="text" name="first_name">
      Прізвище: <input type="text" name="last_name">
      E-Mail: <input type="email" name="user_email">
      <input type="submit" value="Відправити">
    </form>
  </body>
</html>
```

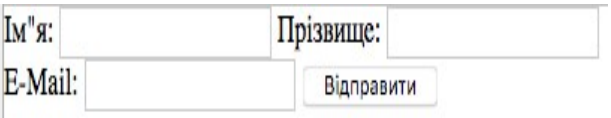


Рис. 8.2. Приклад виконання сценарію.

Основним атрибутом, що визначає тип поля введення, є **`type`**. Якщо атрибут відсутній, значенням за замовчуванням є “text”.

Таблиця 8.2.

Значення атрибуту type

Значення	**Опис**
button	Визначає активну кнопку.
checkbox	Встановлюють прапорці (користувач може вибрати більше одного варіанта із запропонованих).
color	Визначає палітру кольорів (користувач може вибрати значення кольорів у шістнадцятковому форматі).
date	Вказує поле для введення календарної дати.
datetime	Задає поле для введення дати та часу.
datetime-local	Задає поле для введення дати та часу без часового поясу.
e-mail	Вказує поле для введення адреси електронної пошти.
file	Задає елемент керування з кнопкою "Огляд", натиснувши яку можна вибрати і завантажити файл.
hidden	Визначає приховане поле введення. Користувачеві воно не видно.

Продовження Таблиці 8.2.

Значення	**Опис**
image	Вказує, що замість кнопки для надсилання даних на сервер використовується зображення. Шлях до зображення вказується атрибутом src. Також можуть бути використані alt атрибут для вказівки альтернативного тексту, атрибути heigh і width для вказівки висоти і ширини зображення.
month	Визначає поле для вибору місяця, після чого дані вводитимуться у вигляді рік-місяць (наприклад: 2025-01).
number	Визначає поле для введення чисел.
password	Визначає поле для введення пароля (символи, що вводяться, відображаються зірочками, точками або іншими знаками).
radio	Створює радіокнопки (при виборі однієї радіокнопки всі інші стають неактивними).
range	Створює повзунок для вибору чисел у вказаному діапазоні. За замовчуванням діапазон від 0 до 100. Діапазон чисел задається атрибутами min і max.
reset	Визначає кнопку скидання інформації.
search	Створення текстового поля для пошуку.
submit	Створює кнопку надсилання даних форми (кнопка "надіслати").
text	Створює однорядкове текстове поле.
time	Визначає числове поле для введення часу у 24-годинному форматі (наприклад, 13:45).
url	Створює поле для введення URL-адреси.
week	Створює поле для вибору тижня, після чого дані вводяться у форматі рік-тиждень (наприклад: 2024-W25).

Однорядкове текстове поле text

Дозволяє користувачам вводити різноманітну інформацію.

<input type="text" name="ім'я поля" size="розмір" maxlength="макс. кількість символів" value="Текст за замовчуванням">

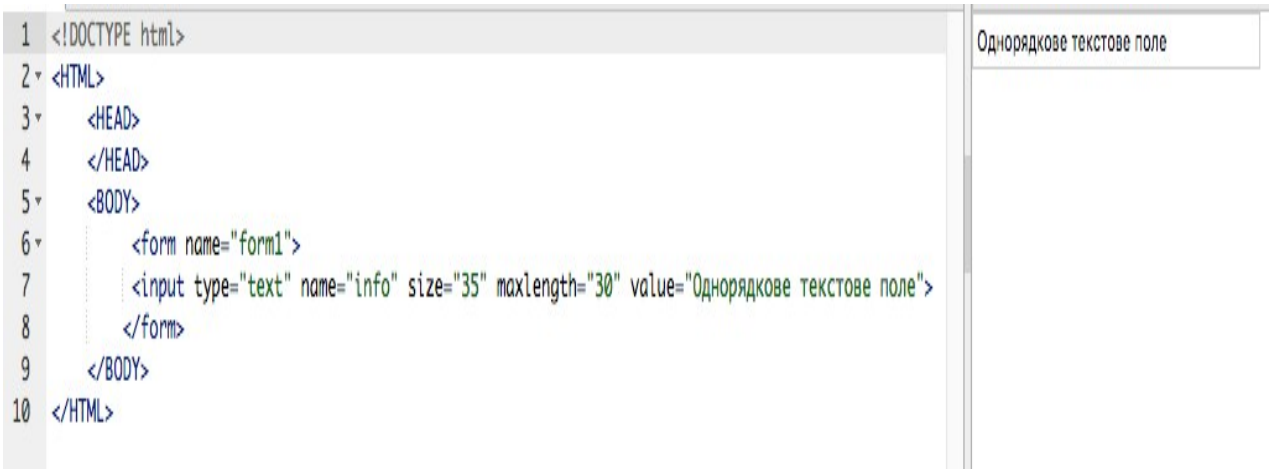
При створенні звичайного текстового поля, що має розмір size та максимальну допустиму довжину maxlength символів, атрибут type набуває значення text.

Якщо вказано параметр value, то поле міститиме (і відображатиме) value-текст.

Під час створення поля слід вказувати ім'я поля, бо цей атрибут є обов'язковим.

Приклад. Форма з однорядковим текстовим полем (див. Рис. 8.3)

```
<form name="form1">
     <input type="text" name="info" size="35" maxlength="30" value="Однорядкове текстове поле">
```




```
</form>
```

Рис. 8.3. Приклад виконання сценарію.

Поле для введення пароля password

Аналогічне текстовому полю, але символи, що набираються користувачем, не відображатимуться на екрані.

<input type="password" name="txtName" size="10" maxlength="5">

Багаторядковий текст

Елемент форми призначений для створення області, де можна вводити кілька рядків тексту. У такому полі можна робити переноси рядків, вони зберігаються при відправленні даних на сервер.

<textarea параметри>
будь-який текст, що буде відображатися всередині поля
</textarea>

Параметри тегу:

- cols - ширина поля у символах.
- disabled - блокує доступ та зміну елемента.
- name - ім'я поля, призначене для того, щоб обробник форми міг його ідентифікувати.
- readonly - встановлює, що поле не може бути змінено користувачем.
- rows - висота поля у рядках тексту.
- wrap - параметри перенесення рядків.

Приклад. Форма з багаторядковим текстом (див. Рис. 8.4)

```
<html>
<body>
<form>
   <textarea name="comment" rows="12" cols="35">Написати коментар автору.</textarea><br>
   <input type="submit" name="submitInfo" value="Надіслати">
</form>
</body>
</html>
```

Рис. 8.4. Приклад виконання сценарію.

Приклад з використанням CSS стилів (див. Рис. 8.5)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок документа</title>
    <style>
      .comment {
        width: 60%;
        height: 100px;
        padding: 10px;
        outline: 0;
        border: 3px solid #1c87c9;
        background: #d0e2bc;
        line-height: 20px;
      }
    </style>
</head>
<body>
    <form>
      <p>Приклад з використанням CSS стилів</p>
                <textarea  class="comment">  Написати  коментар
автору.</textarea>
      <br>
      <input type="submit" name="submitInfo" value="Надіслати">
    </form>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок документа</title>
    <style>
      .comment {
        width: 60%;
        height: 100px;
        padding: 10px;
        outline: 0;
        border: 3px solid #1c87c9;
        background: #d0e2bc;
        line-height: 20px;
      }
    </style>
  </head>
  <body>
    <form>
      <p>Приклад з використанням CSS стилів</p>
      <textarea class="comment"> Написати комментар автору.</textarea>
      <br>
      <input type="submit" name="submitInfo" value="Надіслати">
    </form>
  </body>
</html>
```

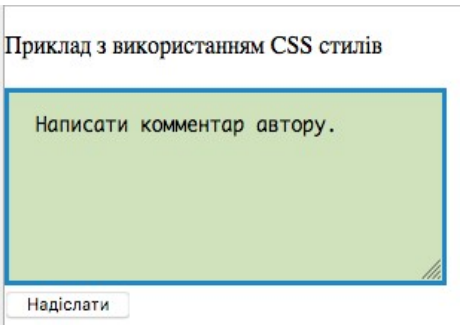


Рис. 8.5. Приклад виконання сценарію.

Приховане поле

Приховане поле не відображається на сторінці та ховає вміст від користувача. Відвідувач не може нічого до нього внести або надрукувати. Мета створення прихованих полів - передавання технічної інформації на сервер. У більшості випадків це необхідно для передавання форм від сторінки до сторінки.

<input type="hidden" параметри>

Список параметрів:

- name – ім'я поля для його ідентифікації обробником форми.
- value – значення поля, яке визначає, яка інформація буде надіслана на сервер.

Кнопки

Кнопки часто застосовуються у формах, особливо при їх відправленні та очищенні. Їх можна створити двома способами – за допомогою тегу <INPUT> та тегу <BUTTON>.

<input type="button" параметри>

Параметри:

- name – ім'я кнопки, призначене для того, щоб обробник форми міг його ідентифікувати.
- value – значення кнопки та одночасно напис на ній.

Спосіб, заснований на використанні тегу <BUTTON>, пропонує розширені можливості створення кнопок. Наприклад, на такій кнопці можна розміщувати будь-які елементи HTML, у тому числі зображення та таблиці.

Приклад. Створення кнопки з малюнком:

```
<button><img src="/images/image.gif" width="10" height="10">
Кнопка з малюнком</button>
```

Таблиця 8.3.

Деякі атрибути кнопок

Атрибут	**Значення**	**Опис**
autofocus	**autofocus**	Вказує браузеру, що кнопка має отримати фокус після завантаження сторінки.
disabled	**disabled**	Деактивує кнопку. (Використовується у випадку, коли кнопка має стати активною після виконання будь-якої дії.)
formaction	**URL**	Задає адресу, куди будуть надсилатися дані форми при натисканні на кнопку. (Використовується лише для кнопок із атрибутом type="submit").
formnovalidate	**formnovalidate**	Скасовує перевірку даних форми на коректність. (Використовується лише для кнопок із атрибутом type="submit").
name	**name**	Визначає назву кнопки.
type	**Визначає тип кнопки.**	
	button	Звичайна кнопка
	reset	Кнопка, яка очищає форму від введених даних
	submit	Кнопка для надсилання даних форми
value	**text**	Встановлює значення кнопки.

Приклад. Використання кнопки (див. Рис. 8.6)

```
<HTML>
          <BODY>
           <h3>Нижче розташована кнопка</h3>
        <form name="form1">
         <button type="button">Натиснути</button>
        </form>
     </BODY>
</HTML>
```

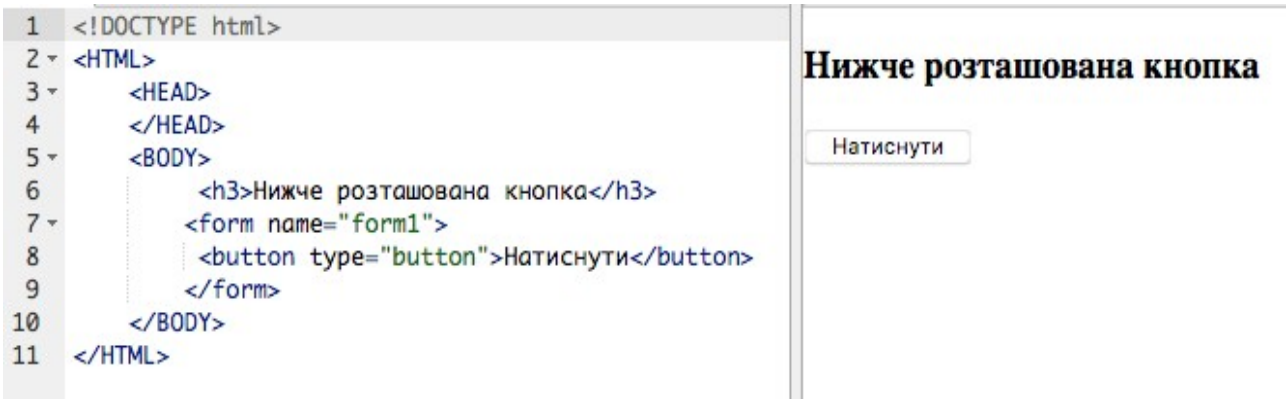


Рис. 8.6. Приклад виконання сценарію.

До тегу <button> можна застосовувати CSS стилі зміни зовнішнього вигляду кнопки, її розміру, кольору, шрифту тексту тощо.

Приклад. (див. Рис. 8.7.)

```
!DOCTYPE html>
<HTML>
     <HEAD>
     </HEAD>
     <BODY>
     Звичайна кнопка
    <button type="button">Додати в кошик</button>
    <hr>
    Кнопка з червоним текстом
       <button type="button" style="color: red;"><b>Підручник з
HTML</b></button>
    <hr>
    Кнопка зі збільшеним розміром шрифту
              <button type="button" style="font: bold 14px
Arial;">Завантажити книгу </button>
     </BODY>
</HTML>
```

Рис. 8.7. Приклад виконання сценарію.

Кнопка SUBMIT

<input type="submit" параметри>

Служить для надсилання форми сценарію. При створенні необхідно вказати 2 атрибути:

type="submit"

value="Текст кнопки".

Атрибут name необхідний, якщо кнопок кілька, і всі вони створені для різних операцій (наприклад, кнопки «Зберегти», «Видалити», «Редагувати» тощо) Після натискання на кнопку сценарію передається рядок ім'я = текст кнопки.

<input type="submit" name="Ім'я кнопки" value="Текст кнопки">

Кнопка RESET

При натисканні на кнопку RESET дані форми повертаються в початкове значення.

Як правило, цю кнопку застосовують для очищення інформації, введеної в поля форми.

Для великих форм доцільно відмовитись від використання кнопки RESET, щоб помилково на неї не натиснути, адже тоді доведеться заповнювати форму заново.

<input type="reset" параметри>

Прапорці

Прапорці checkbox пропонують користувачеві низку варіантів і дозволяють вибрати кілька з них. Група прапорців складається з елементів <input>, що мають атрибути name та type (checkbox). Щоб елемент був вибраний за замовчуванням, слід позначити його як checked. Якщо елемент вибраний, то сценарію надійде рядок ім'я=значення, інакше в обробник форми нічого не прийде. Тобто невибрані прапорці взагалі не виявляють себе в переданому наборі даних.

Приклад. Фрагмент документа (Див. Рис. 8.8):

```
<input type="checkbox" name="option1" value="a1" checked> Кава<br>
<input type="checkbox" name="option2" value="a2"> Чай<br>
<input type="checkbox" name="option3" value="a3"> Сік<br>
<input type="checkbox" name="option4" value="a4" checked>
Молоко<br>
```

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Заголовок</title>
5    </head>
6    <body>
7
8     <form name="form1" method="post" action="../admin/add.php">
9     <input type="checkbox" name="option1" value="a1" checked> Кава<br>
10    <input type="checkbox" name="option2" value="a2"> Чай<br>
11    <input type="checkbox" name="option3" value="a3"> Сік<br>
12    <input type="checkbox" name="option4" value="a4" checked> Молоко<br>
13    </form>
14
15 </body>
16 </html>
```

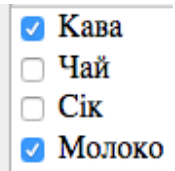


Рис. 8.8. Приклад виконання сценарію.

Приклад загального коду сторінки.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
  </head>
  <body>
   <form name="form1" method="post" action="../admin/add.php">
```

```
      <input  type="checkbox"  name="option1"  value="a1"  checked>
Кава<br>
   <input type="checkbox" name="option2" value="a2"> Чай<br>
   <input type="checkbox" name="option3" value="a3"> Сік<br>
      <input  type="checkbox"  name="option4"  value="a4"  checked>
Молоко<br>
   </form>
 </body>
</html>
```

Перемикачі

Перемикачі використовують, коли необхідно вибрати один варіант з декількох запропонованих.

<input type="radio" name="ім'я" параметри>

Параметри:

- checked - попередній вибір перемикача. За визначенням, набір перемикачів може мати лише один вибраний пункт. У будь-якому випадку, буде відмічено елемент, що знаходиться в коді HTML останнім.
- name - ім'я групи перемикачів для ідентифікації поля. Оскільки перемикачі є груповими елементами, ім'я у всіх елементів групи має бути однаковим.
- value - визначає, яке значення буде надіслано на сервер. Тут кожен елемент повинен мати своє унікальне значення, щоб можна було ідентифікувати, який пункт був вибраний користувачем.

Приклад фрагменту документа (див. Рис. 8.9):

```
<input name="mydrink" type="radio" value="coffee"> Кава <br>
<input name="mydrink" type="radio" value="tea"> Чай <br>
<input  name="mydrink"  type="radio"  value="juice"  checked>  Сік
(вибраний за замовчуванням)<br>
```

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Заголовок</title>
5    </head>
6    <body>
7
8     <form name="form1">
9      <input name="mydrink" type="radio" value="coffee"> Кава <br>
10     <input name="mydrink" type="radio" value="tea"> Чай <br>
11     <input name="mydrink" type="radio" value="juice" checked> Сік <br>
12    </form>
13
14 </body>
15 </html>
```

Кава
Чай
Сік

Рис. 8.9. Приклад виконання сценарію.

Приклад загального коду сторінки.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
```

```
  </head>
  <body>
   <form name="form1">
    <input name="mydrink" type="radio" value="coffee"> Кава <br>
    <input name="mydrink" type="radio" value="tea"> Чай <br>
     <input name="mydrink" type="radio" value="juice" checked> Сік
<br>
   </form>
</body>
</html>
```

Поле зі списком

```
<select параметри>
<option параметри>Пункт 1</option>
<option>Пункт 2</option>
<option>Пункт 3</option>
</select>
```

Тег <SELECT> дозволяє створити елемент інтерфейсу у вигляді списку, що розкривається, а також список з одним або множинним вибором.

Вигляд списку залежить від використання параметра size тегу <SELECT>, який встановлює висоту списку.

Кожен пункт створюється за допомогою тегу <OPTION>, який має бути вкладений у контейнер <SELECT>. Ширина списку визначається найширшим текстом, вказаним у тегу <OPTION>, а також може змінюватись за допомогою стилів.

Параметри тегу <SELECT>, за допомогою яких можна змінювати вигляд та подання списку.

- multiple - повідомляє браузеру відображати вміст <SELECT> як список множинного вибору. Кінцевий вигляд списку залежить від параметра size. Якщо він відсутній, то висота списку дорівнює кількості пунктів, якщо значення size менше числа пунктів, то з'являється вертикальна смуга прокручування.
- name - визначає унікальне ім'я елемента <SELECT>.
- size - встановлює висоту списку. Якщо значення параметра size дорівнює одиниці, то список перетворюється на розкривний. При додаванні параметра multiple до тегу <SELECT> при size=1, список відображається як «крутилка». В інших випадках виходить список з одним або множинним вибором. Стандартне значення залежить від параметра multiple. Якщо він присутній, то розмір списку дорівнює кількості елементів. Коли параметра multiple немає, значення значення size дорівнює 1.

  Параметри <OPTION>

- selected - робить поточний пункт списку вибраним. Якщо до тегу <SELECT> додано параметр multiple, можна вибирати більше одного пункту.

- value - визначає значення списку, яке буде надіслано на сервер. На сервер відправляється пара "ім'я/значення", де ім'я задається параметром name тегу <SELECT>, а значення - параметром value вибраних пунктів списку. Значення може як збігатися з текстом пункту, так і бути самостійним.

Приклад (Див. Рис. 8.10).

```
<!DOCTYPE html>
<HTML>
     <HEAD>
     </HEAD>
     <BODY>
     <form action="action_form.php" target="_top">
      <select>
        <option value="books">Computer Science</option>
        <option value="html">HTML</option>
        <option value="css">CSS</option>
        <option value="php">PHP</option>
        <option value="js">JavaScript</option>
      </select>
    </form>
     </BODY>
</HTML>
```

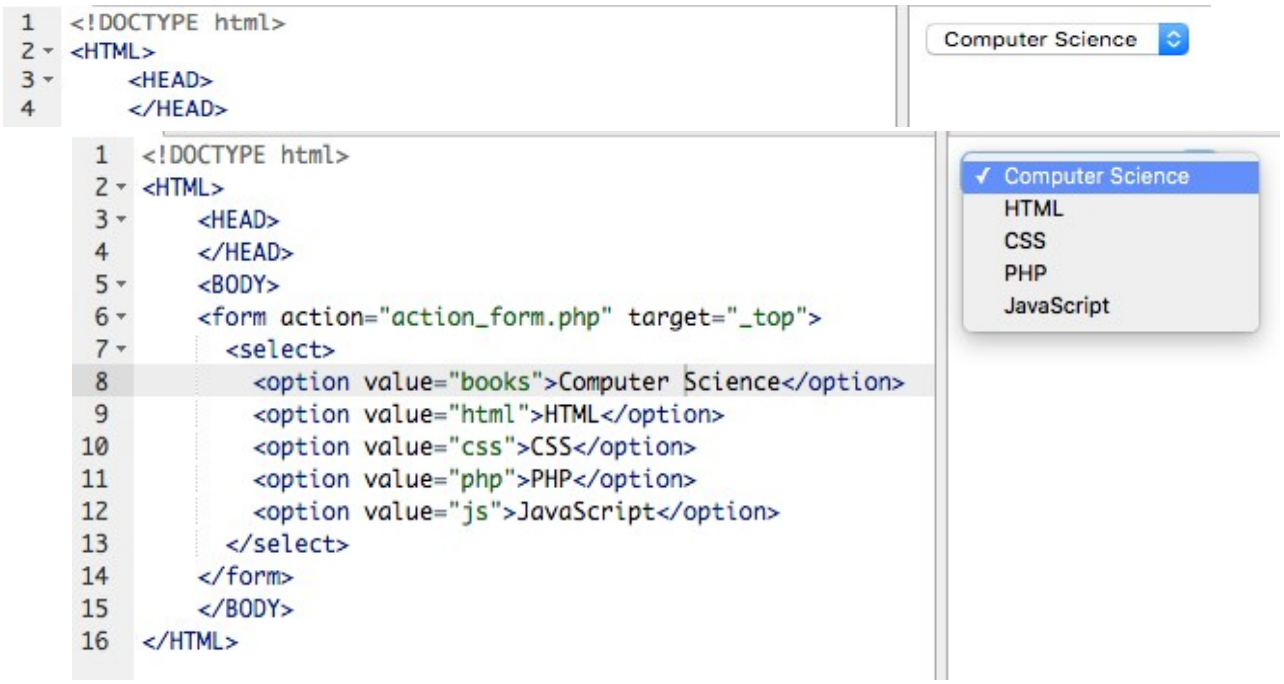


Рис. 8.10. Приклад виконання сценарію.

Поле із зображенням

Поле із зображенням аналогічне за дією кнопці SUBMIT, але є малюнком. Це розширює можливості оформлення форми. Коли користувач натискає на малюнок, дані форми відправляються на сервер і обробляються програмою, заданою параметром action тегу <FORM>.

<input type="image" параметри>

Параметри:

- align - визначає вирівнювання зображення.
- alt - альтернативний текст для кнопки із зображенням.
- border - товщина рамки навколо зображення пікселів.
- name - ім'я поля для звернення до нього зі скриптів або отримання значення обробником форми.

Хоча за своїм призначенням вказане поле схоже на кнопку SUBMIT, його параметри збігаються з параметрами тегу <IMG>, який додає зображення на вебсторінку.

Приклад. Див. Рис.8.11.

```
<!DOCTYPE html>
<html>
<head>
  <title>Заголовок</title>
</head>
<body>

<form>
 <input type="image" name="button_image" border="1px" src="1.jpg"
width="70" height="70">
</form>
</body>
</html>
```

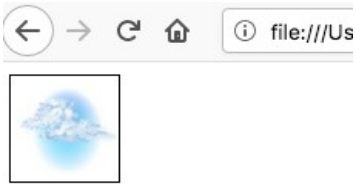


Рис. 8.11. Приклад виконання сценарію.

Мітка

Тег **<label>** визначає текстову мітку (позначку) для тегу <input>. Мітка є звичайним текстом, клацнувши по якому, користувач може вибрати елемент форми. Вона полегшує використання форми, так як у елементи форми не завжди зручно потрапити курсором.

Тег <label> парний.

Тег <label> також використовується для визначення гарячих клавіш на клавіатурі та переходу на активний елемент, подібно до посилань.

Зв'язати текстову мітку та форму, до якої вона відноситься, можна так:

1. Встановити ідентифікатор id всередині елемента <input> і вказати його ім'я як атрибут для тегу <label>.

Приклад. Див. Рис. 8.12.

```
<body>
    <form>
      <label for="lfname">Ім'я користувача:</label>
      <input id="lfname" name="fname" type="text">
    </form>
</body>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
  </head>
  <body>
    <form>
      <label for="lfname">Ім'я користувача:</label>
      <input id="lfname" name="fname" type="text">
    </form>
  </body>
</html>
```

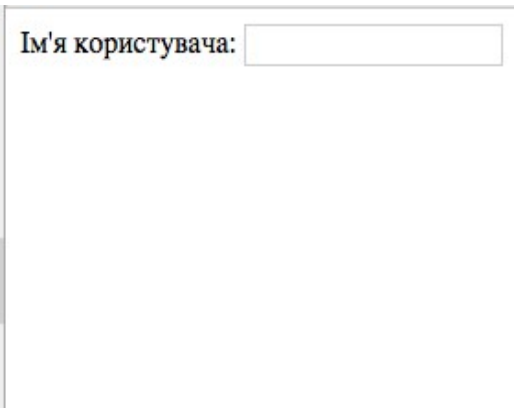


Рис. 8.12. Приклад виконання сценарію.

2. Помістити <input> в елемент <label>.

Приклад. Див. Рис. 8.13.

```
<body>
    <form>
      <label>Ім'я користувача
      <input id="User" name="Ім'я" type="text">
      </label>
    </form>
  </body>
```

```
<!DOCTYPE html>
<html>
   <head>
      <title>Заголовок</title>
   </head>
  <body>
    <form>
      <label>Ім'я користувача
      <input id="User" name="Ім'я" type="text">
      </label>
    </form>
  </body>
</html>
```

Рис. 8.13. Приклад виконання сценарію.

Для стилізації тегу <label> можна використати властивості CSS font

Приклад. Див. Рис. 8.14.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
      body {
       padding: 20px;
      }
      label {
        font-size: 20px;
        font-weight: 700;
        color: #1c87c9;
      }
```

```
input {
        width: 50%;
        height: 28px;
        padding: 4px 10px;
        border: 1px solid #777;
        background: #cce6ff;
        color: #1c87c9;
        font-size: 16px;
      }
    </style>
  </head>
  <body>
    <form>
      <label>Ваше Ім'я:</label>
      <input id="User" name="Name" type="text"/>
    </form>
  </body>
</html>
```

Рис. 8.14. Приклад виконання сценарію.

Легенда

Парний тег **<legend>** задає заголовок елементам форми, згрупованих тегом <fieldset>.

Група елементів форми у браузері виділяються рамкою, а вміст тега <legend> вставляється у цю рамку.

Приклад. Див. Рис. 8.15.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
  </head>
  <body>
    <form>
```

```
      <fieldset>
        <legend>Персональні дані:</legend>
        Ім'я: <input type="text"><br>
        Ел.пошта: <input type="text"><br>
        Дата народження: <input type="text"><br>
        Місце народження: <input type="text">
      </fieldset>
    </form>
  </body>
</html>
```

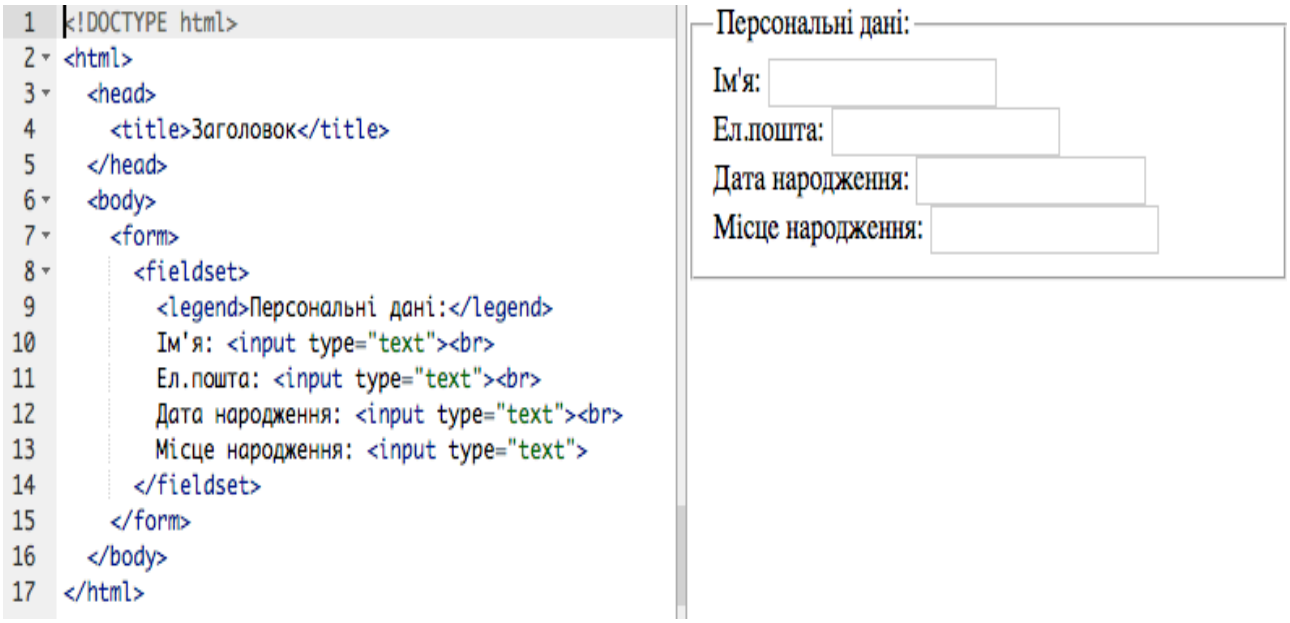


Рис. 8.15. Приклад виконання сценарію.

Валідація форм на боці клієнта

На сайтах з формами реєстрації можна помітити, що під час введення даних у неправильному форматі користувачеві відразу повідомляють про проблему.

У міру введення браузер та/або сервер перевіряють дані, щоб визначити, чи відповідають вони потрібному формату. Валідація (перевірка), що виконується в браузері, називається валідацією на стороні клієнта, а на сервері - валідацією на стороні сервера.

Валідація на стороні клієнта - це первинна перевірка введених даних, яка покращує зручність взаємодії з інтерфейсом, виявляє некоректні дані на стороні клієнта і дозволяє користувачеві негайно їх виправити.

Якщо формат коректний, програма дозволяє відправити дані на сервер і (зазвичай) зберегти в базі даних; в іншому випадку виводиться повідомлення про те, що потрібно виправити, дозволяючи знову ввести дані.

Типи валідації на стороні клієнта:

- Вбудована валідація форм використовує функціонал валідації HTML5. HTML5-валідація зазвичай не вимагає великої кількості JavaScript-коду і демонструє кращу продуктивність, але не настільки налаштовувана, як валідація за допомогою JavaScript.
- JavaScript-валідація кодується за допомогою JavaScript. Вона повністю налаштовується, але вимагає програмування всієї логіки (або використання бібліотеки).

Однією з найважливіших функцій елементів форм HTML5 є здатність валідувати більшу частину даних користувача без використання JavaScript. Це виконується за допомогою *атрибутів валідації у елементів форми*:

- required: Визначає, що для надсилання форми це поле має бути попередньо заповнене.
- minlength та maxlength: Задає мінімальну та максимальну довжину текстових даних (рядків).
- min і max: Задає мінімальне та максимальне значення для поля, розрахованого на числовий тип даних.
- type: Визначає тип даних, на який розраховано поле: число, email-адресу або якийсь інший встановлений тип.
- pattern: За допомогою регулярного виразу визначає шаблон, якому повинні відповідати дані, що вводяться.

Якщо дані, введені в поле форми, відповідають правилам перерахованих вище атрибутів, вони вважаються валідними, якщо ні – не валідними.

Матеріали для самоосвіти:

https://css.in.ua/html/tag/form
https://w3schoolsua.github.io/html/html_forms.html
http://sites.znu.edu.ua/webprog/lect/1215.ukr.html
https://cpto.dp.ua/public_html/posibnyky/basic_html/urok13_1.html
https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

9. ПРАКТИЧНІ РОБОТИ
Практична робота №1
Основи алгоритмізації мовою JavaScript

Завдання 1

Написати код мовою JavaScript для розв'язування задачі:

Запитати у користувача ім'я, вік та номер курсу. Після цього вивести вікно з повідомленням "Вітаю, [Ім'я] !", потім – вікно з повідомленням "Ви – студент!", якщо введено номер курсу від 1 до 6 або "Ви – не студент..." в іншому випадку.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.1.1):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
          <script>
          let name=prompt("Введіть Ваше ім_я:"," ");
          let age=prompt("Введіть Ваш вік:"," ");
          let course=prompt("Введіть Ваш номер курсу:"," ");
          let isStudent = course>=1 && course<=6;
          alert(`Вітаю, ${name}!`);
            isStudent ? alert("Ви - студент!") : alert("Ви – не
студент...");
          </script>

     </HEAD>
     <BODY>
          Практична робота. Завдання 1
     </BODY>
</HTML>
```

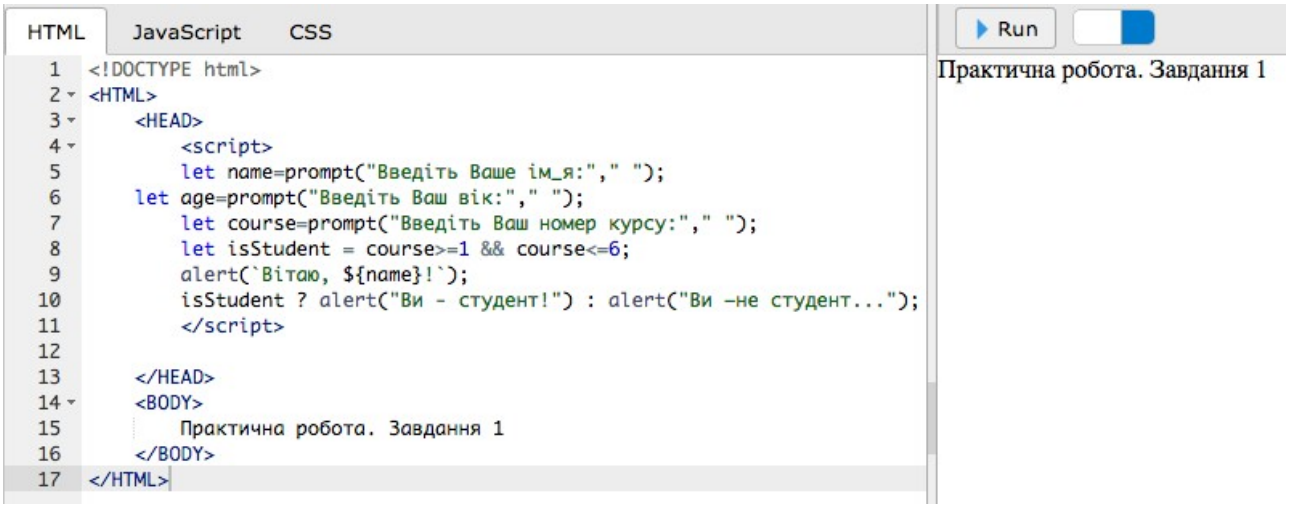


Рис. 9.1.1. Виконання Завдання 1.

Завдання 2

Написати код мовою JavaScript для розв'язування задачі:

Ввести цілі числа M і N (M<N). Обчислити суму квадратів чисел від M до N.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.1.2):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
         <script>
         let M=prompt("введіть ціле число M:",1);
         let N=prompt("введіть ціле число N>M:",10);
         let s=0;
         for (let i=M; i<=N;i++) {
           s+=i*i;
         }
         alert(`Сума квадратів чисел від ${M} до ${N} = ${s}`);
         </script>
     </HEAD>
     <BODY>
          Практична робота. Завдання 2
     </BODY>
</HTML>
```

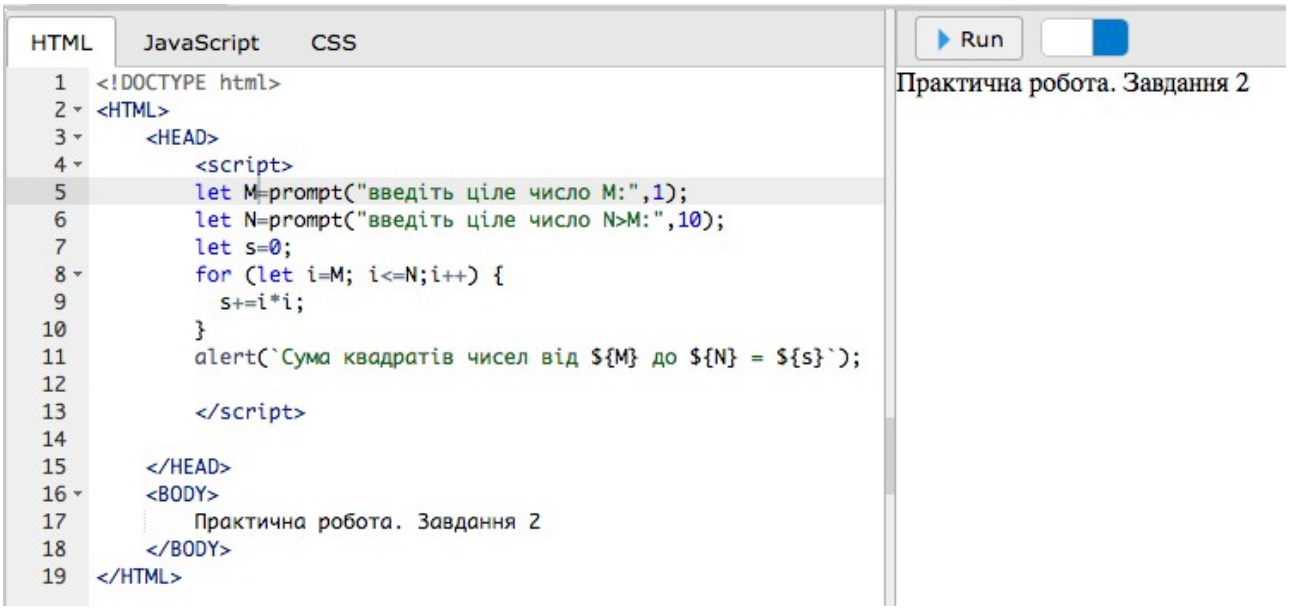


Рис. 9.1.2. Виконання Завдання 2.

Завдання 3

Написати код мовою JavaScript для розв'язування задачі:

Написати функцію, за якою для двох заданих користувачем чисел M і N (M<N) виводяться квадрати чисел від M до N.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.1.3):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
         <script>
         function squares(a,b) {
          for (let i=a; i<=b; i++) {
          console.log(i*i)
          }
         }
          let M=prompt("введіть число M:",0);
          let N=prompt("введіть число N > M:",10);
squares(M,N);

</script>

     </HEAD>
     <BODY>
          Практична робота. Завдання 3
     </BODY>
</HTML>
```

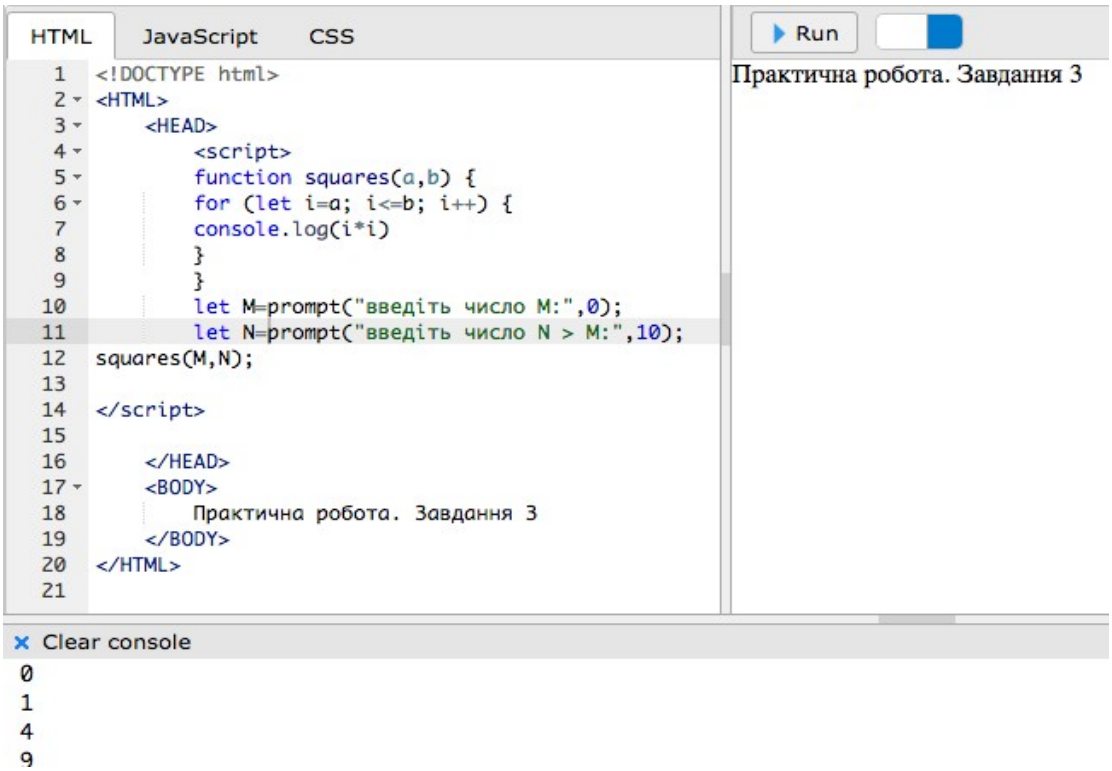


Рис. 9.1.3. Виконання Завдання 3.

Завдання 4

Написати код мовою JavaScript для розв'язування задачі:

Ввести ціле число N. Сформувати масив з N випадкових цілих чисел від 0 до 100. У випадку введення N>20 встановити N=20.

Вивести сформований масив у вікно браузера (document.write()).

Вивести в консоль браузера: всі елементи масиву, що кратні 5, але не кратні 10; суму цих елементів.

Код вебсторінки, відповідної розв’язку, може бути таким (див. Рис. 9.1.4):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
         <script>
       let arr=[];
         let n = prompt("Введіть кількість елементів масиву:",10);
if (n>20) {
n=20;
}
        for (let i=0; i<n; i++) {
          arr[i]=Math.round(100*Math.random());
            }
        document.write('Масив: '+arr);
        let s=0;
         for (let i=0; i<n; i++) {
          if (arr[i]%5==0 && arr[i]%10!= 0) {
               console.log(arr[i]);
               s+=arr[i];
          }
            }
         console.log('s = '+s);
         </script>

     </HEAD>
     <BODY>
          <p> Практична робота. Завдання 4 </p>
     </BODY>
</HTML>
```

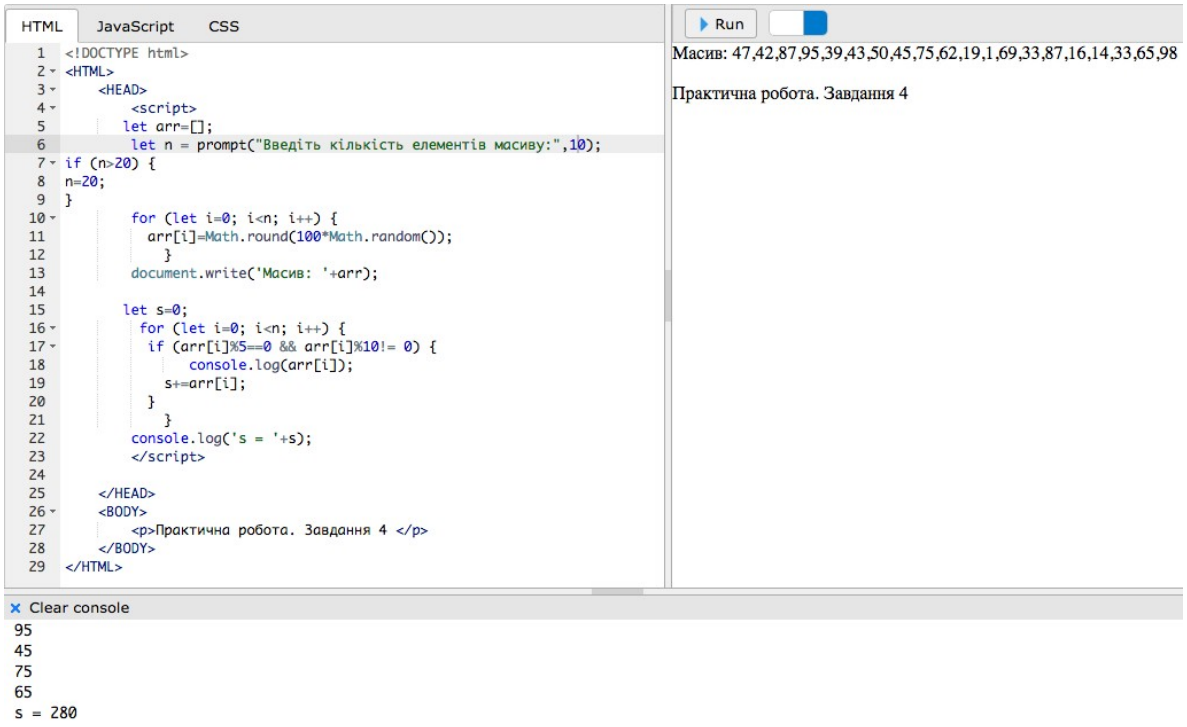


Рис. 9.1.4. Виконання Завдання 4.

Завдання 5

Написати код мовою JavaScript для розв'язування задачі:

Ввести ціле число N. Сформувати рядок з N випадкових малих українських літер.

Вивести сформований рядок у консоль браузера.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.1.5):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
         <script>
         let st="йцукенгшщзхїфівапролджєячсмитьбю";

         let len =prompt("введіть довжину рядка:",10);
         let result="";
         for (let i=0; i<len; i++) {
          ind=Math.round((st.length-1)*Math.random());
          result=result+st[ind];
          }
        console.log('result: '+result);

        </script>
     </HEAD>
     <BODY>
          <p>Практична робота. Завдання 5</p>
     </BODY>
</HTML>
```

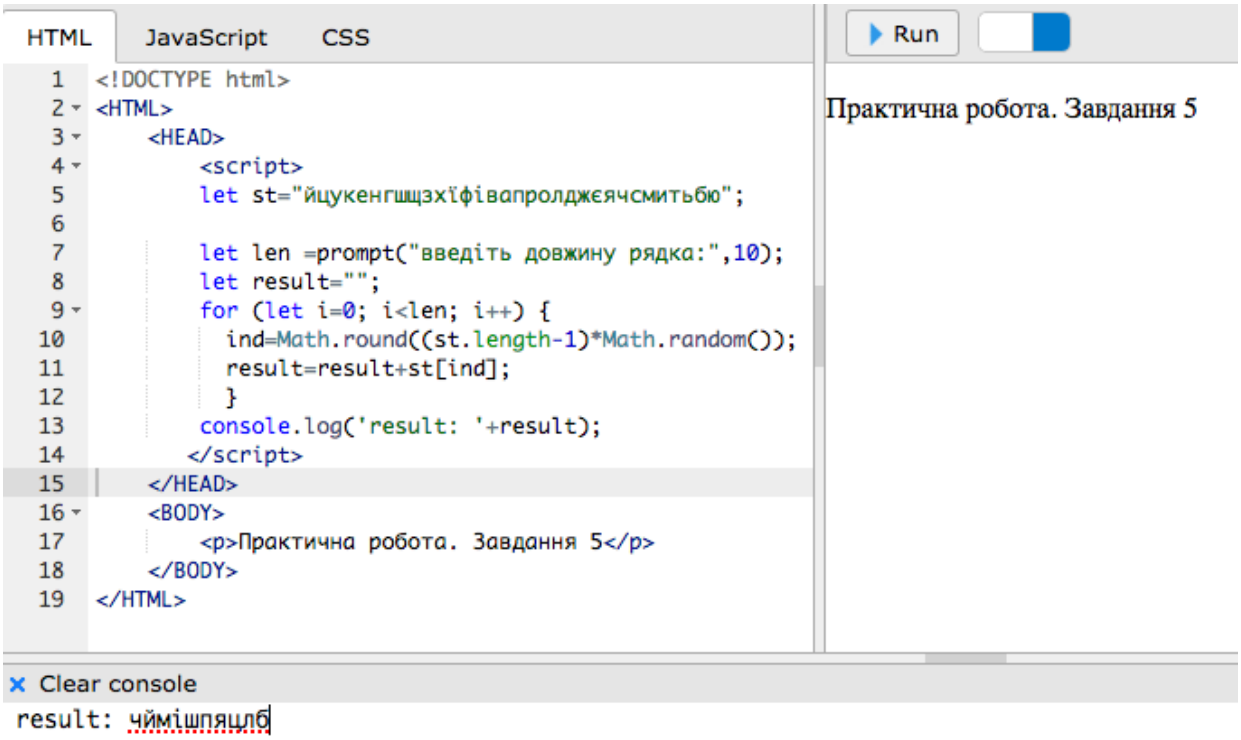


Рис. 9.1.5. Виконання Завдання 5.

Практична робота №2
Робота з об'єктною моделлю документа. Маніпуляції стилями CSS. Планування викликів функцій

Завдання 1.

Створити HTML-документ з таким вмістом:
три абзаци тексту (<p>), нумерований список (<ol>) з семи елементів, чотири блоки ( ).

Створити таблицю стилів документу, в якій описати
а) три селектори-ідентифікатори з визначеними параметрами:
вирівнювання тексту за лівою межею, колір фону жовтий;
вирівнювання тексту за центром, колір фону блакитний;
вирівнювання тексту за правою межею, колір фону рожевий;
б) чотири селектори-класи з визначеними параметрами:
вирівнювання тексту за лівою межею, розмір шрифту 12 пунктів, колір шрифту червоний;
вирівнювання тексту за лівою межею, розмір шрифту 14 пунктів, колір шрифту зелений;
вирівнювання тексту за лівою межею, розмір шрифту 16 пунктів, колір шрифту синій;
вирівнювання тексту за лівою межею, розмір шрифту 18 пунктів, колір шрифту сірий;
в)селектор-клас з визначеними параметрами: розмір шрифту 14 пунктів, колір шрифту червоний, колір фону світло-сірий.

Встановити форматування трьох абзаців тексту за відповідними селекторами-ідентифікаторами а)

Встановити форматування блоків за відповідними селекторами-класами б)

Встановити форматування кожного елемента списку за селектором-класом в)

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.2.1):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
      <style>

        #p1 {text-align: left; background-color: Yellow;}
        #p2 {text-align: center; background-color: LightBlue;}
        #p3 {text-align: right; background-color: Pink;}
        .d1 {text-align: left; font-size: 12pt; color: #FF0000;}
        .d2 {text-align: left; font-size: 14pt; color: #00FF00;}
        .d3 {text-align: left; font-size: 16pt; color: #0000FF;}
        .d4 {text-align: left; font-size: 18pt; color: #777777;}
        .l0 {font-size: 14pt; color: Red;background-color: LightGray;}
     </style>
     </HEAD>
     <BODY>
          <p id="p1">Цей абзац має ідентифікатор p1</p>
          <p id="p2">Цей абзац має ідентифікатор p2</p>
          <p id="p3">Цей абзац має ідентифікатор p3</p>

          <ol>
        <li class="l0">Елемент списку 1</li>
        <li class="l0">Елемент списку 2</li>
        <li class="l0">Елемент списку 3</li>
        <li class="l0">Елемент списку 4</li>
```

```
        <li class="l0">Елемент списку 5</li>
        <li class="l0">Елемент списку 6</li>
        <li class="l0">Елемент списку 7</li>
       </ol>
        <div class="d1">Блок 1</div>
        <div class="d2">Блок 2</div>
        <div class="d3">Блок 3</div>
        <div class="d4">Блок 4</div>

    </BODY>
</HTML>
```

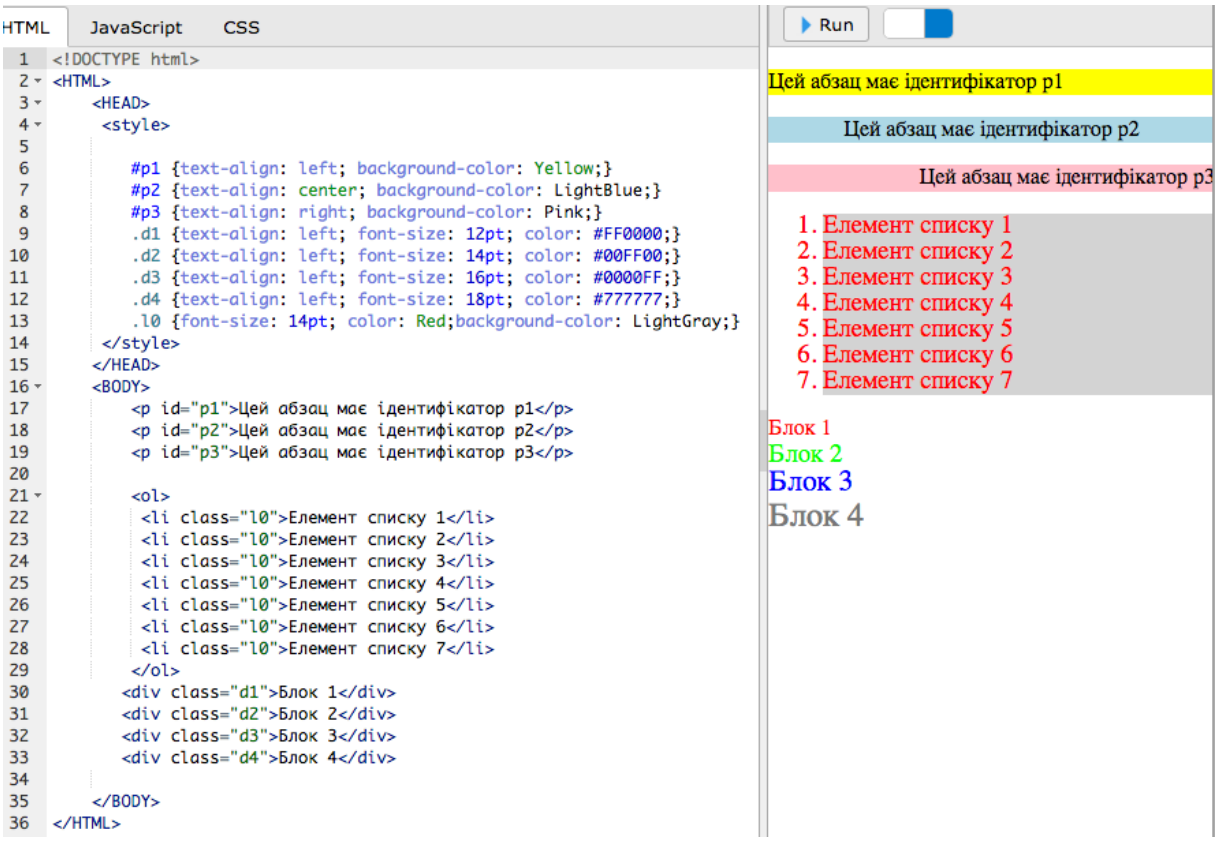


Рис. 9.2.1. Виконання Завдання 1.

Зауваження. *Завдання 2, 3, 4 виконуються з елементами документа, створеного у завданні 1.*

Завдання 2 Написати код на JavaScript, за яким при натисненні лівої кнопки миші на абзаці з ідентифікатором p2 буде виводитись вікно з текстом цього абзацу.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.2.2):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
      <style>

        #p1 {text-align: left; background-color: Yellow;}
        #p2 {text-align: center; background-color: LightBlue;}
        #p3 {text-align: right; background-color: Pink;}
        .d1 {text-align: left; font-size: 12pt; color: #FF0000;}
        .d2 {text-align: left; font-size: 14pt; color: #00FF00;}
        .d3 {text-align: left; font-size: 16pt; color: #0000FF;}
        .d4 {text-align: left; font-size: 18pt; color: #777777;}
        .l0 {font-size: 14pt; color: Red;background-color: LightGray;}
     </style>
         <script>
          function info1(){
            let elem=document.getElementById('p2');
            alert(elem.innerHTML);
          }
         </script>
      </HEAD>
      <BODY>
           <p id="p1">Цей абзац має ідентифікатор p1</p>
           <p id="p2" onClick='info1()'>Цей абзац має ідентифікатор
p2</p>
           <p id="p3">Цей абзац має ідентифікатор p3</p>

           <ol>
          <li class="l0">Елемент списку 1</li>
          <li class="l0">Елемент списку 2</li>
          <li class="l0">Елемент списку 3</li>
          <li class="l0">Елемент списку 4</li>
          <li class="l0">Елемент списку 5</li>
          <li class="l0">Елемент списку 6</li>
          <li class="l0">Елемент списку 7</li>
         </ol>
         <div class="d1">Блок 1</div>
         <div class="d2">Блок 2</div>
         <div class="d3">Блок 3</div>
         <div class="d4">Блок 4</div>

     </BODY>
</HTML>
```

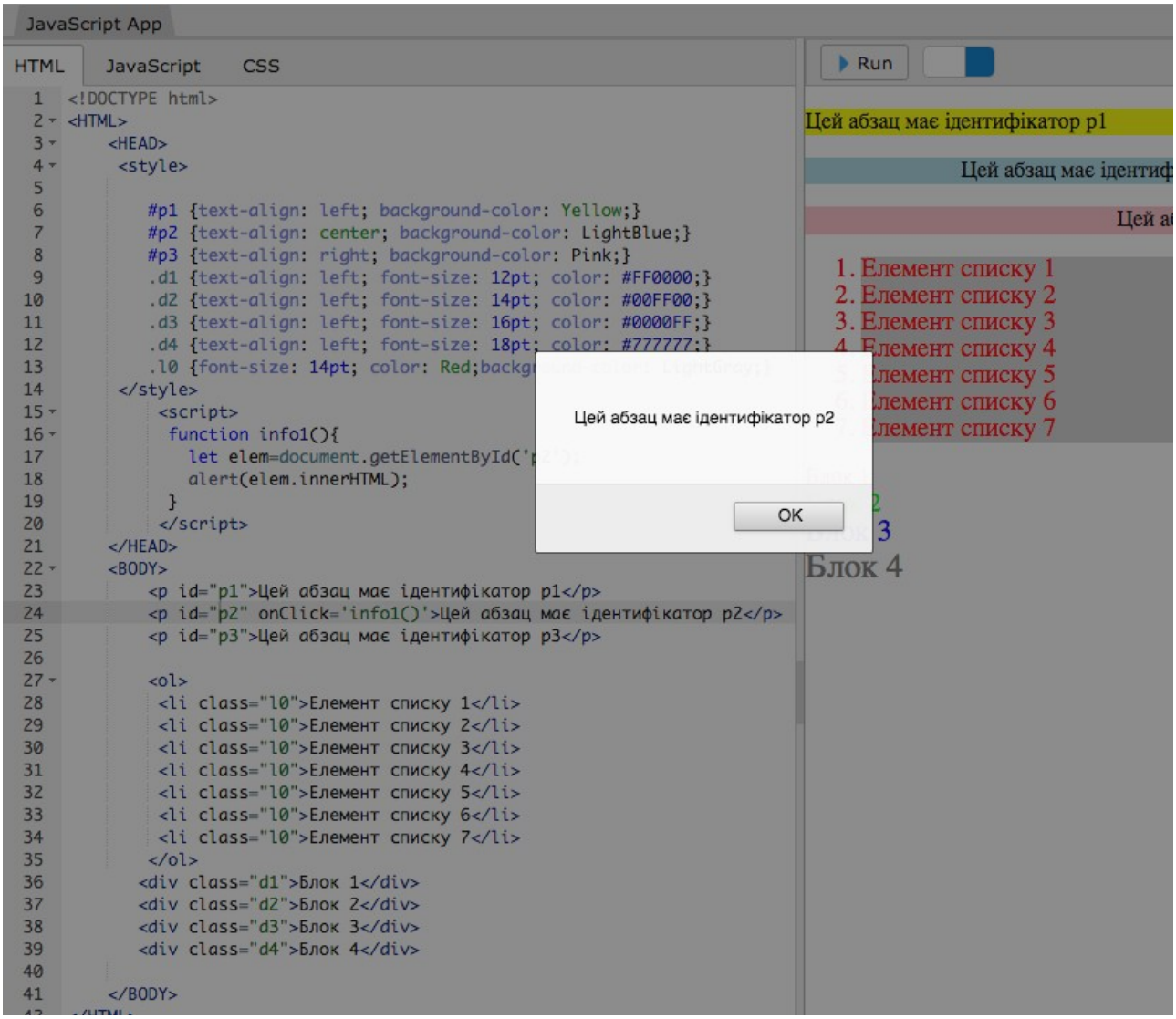


Рис. 9.2.2. Виконання Завдання 2.

Завдання 3. Написати код на JavaScript, за яким при натисненні лівої кнопки миші на останньому з абзаців колір тексту першого абзацу буде змінено на червоний.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.2.3):

```
 <!DOCTYPE html>
<HTML>
     <HEAD>
      <style>

        #p1 {text-align: left; background-color: Yellow;}
        #p2 {text-align: center; background-color: LightBlue;}
        #p3 {text-align: right; background-color: Pink;}
        .d1 {text-align: left; font-size: 12pt; color: #FF0000;}
        .d2 {text-align: left; font-size: 14pt; color: #00FF00;}
        .d3 {text-align: left; font-size: 16pt; color: #0000FF;}
        .d4 {text-align: left; font-size: 18pt; color: #777777;}
        .l0 {font-size: 14pt; color: Red;background-color: LightGray;}
    </style>
        <script>
```

```
          function info1(){
            let elem=document.getElementById('p2');
            alert(elem.innerHTML);
          }
          function info2(){
            let elem=document.getElementsByTagName('p');
            elem[0].style.color="Red";
          }
        </script>
      </HEAD>
      <BODY>
           <p id="p1">Цей абзац має ідентифікатор p1</p>
           <p id="p2" onClick='info1()'>Цей абзац має ідентифікатор
p2</p>
           <p id="p3" onClick="info2()">Цей абзац має ідентифікатор
p3</p>

           <ol>
          <li class="l0">Елемент списку 1</li>
          <li class="l0">Елемент списку 2</li>
          <li class="l0">Елемент списку 3</li>
          <li class="l0">Елемент списку 4</li>
          <li class="l0">Елемент списку 5</li>
          <li class="l0">Елемент списку 6</li>
          <li class="l0">Елемент списку 7</li>
        </ol>
         <div class="d1">Блок 1</div>
         <div class="d2">Блок 2</div>
         <div class="d3">Блок 3</div>
         <div class="d4">Блок 4</div>

      </BODY>
</HTML>
```

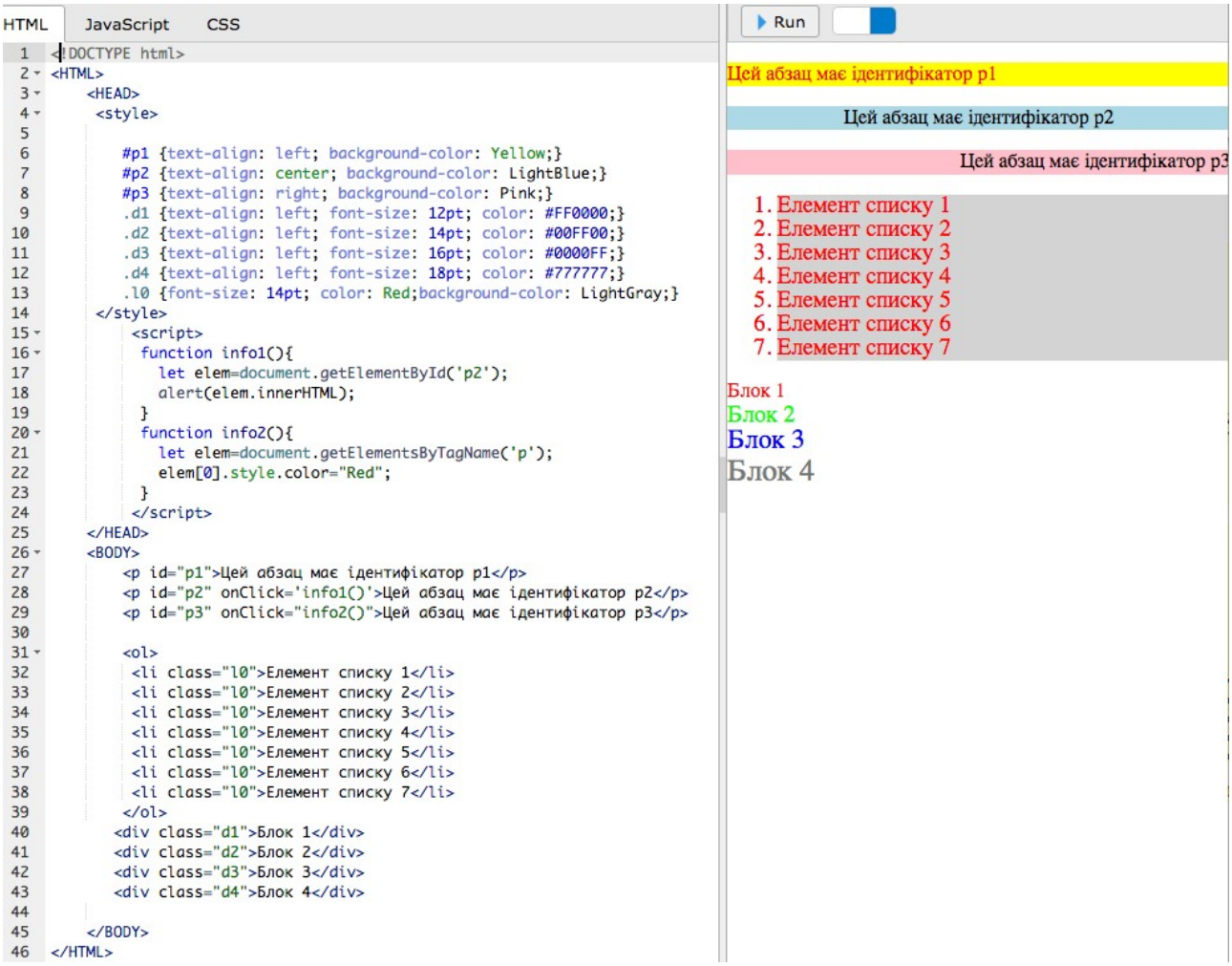


Рис. 9.2.3. Виконання Завдання 3.

Завдання 4. Написати код на JavaScript, за яким при натисненні лівої кнопки миші на першому з блоків форматування першого і останнього блоків будуть мінятись місцями з інтервалом 1 секунду.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.2.4):

```
<!DOCTYPE html>
<HTML>
     <HEAD>
      <style>

        #p1 {text-align: left; background-color: Yellow;}
        #p2 {text-align: center; background-color: LightBlue;}
        #p3 {text-align: right; background-color: Pink;}
        .d1 {text-align: left; font-size: 12pt; color: #FF0000;}
        .d2 {text-align: left; font-size: 14pt; color: #00FF00;}
        .d3 {text-align: left; font-size: 16pt; color: #0000FF;}
        .d4 {text-align: left; font-size: 18pt; color: #777777;}
        .l0 {font-size: 14pt; color: Red;background-color: LightGray;}
    </style>
        <script>
         function info1(){
           let elem=document.getElementById('p2');
           alert(elem.innerHTML);
```

```
          }
          function info2(){
            let elem=document.getElementsByTagName('p');
            elem[0].style.color="Red";
          }
          function change(d){
            let first=0;
            let last=d.length-1;
            let tmp = d[first].className;
            d[first].className = d[last].className;
            d[last].className = tmp;
           }

         function divs(){
           let d = document.getElementsByTagName('div');
           setInterval(change,1000,d);
      }
         </script>
      </HEAD>
      <BODY>
           <p id="p1">Цей абзац має ідентифікатор p1</p>
           <p id="p2" onClick='info1()'>Цей абзац має ідентифікатор
p2</p>
           <p id="p3" onClick="info2()">Цей абзац має ідентифікатор
p3</p>

           <ol>
         <li class="l0">Елемент списку 1</li>
         <li class="l0">Елемент списку 2</li>
         <li class="l0">Елемент списку 3</li>
         <li class="l0">Елемент списку 4</li>
         <li class="l0">Елемент списку 5</li>
         <li class="l0">Елемент списку 6</li>
         <li class="l0">Елемент списку 7</li>
        </ol>
         <div class="d1" onClick="divs()">Блок 1</div>
         <div class="d2">Блок 2</div>
         <div class="d3">Блок 3</div>
         <div class="d4">Блок 4</div>

      </BODY>
</HTML>
```

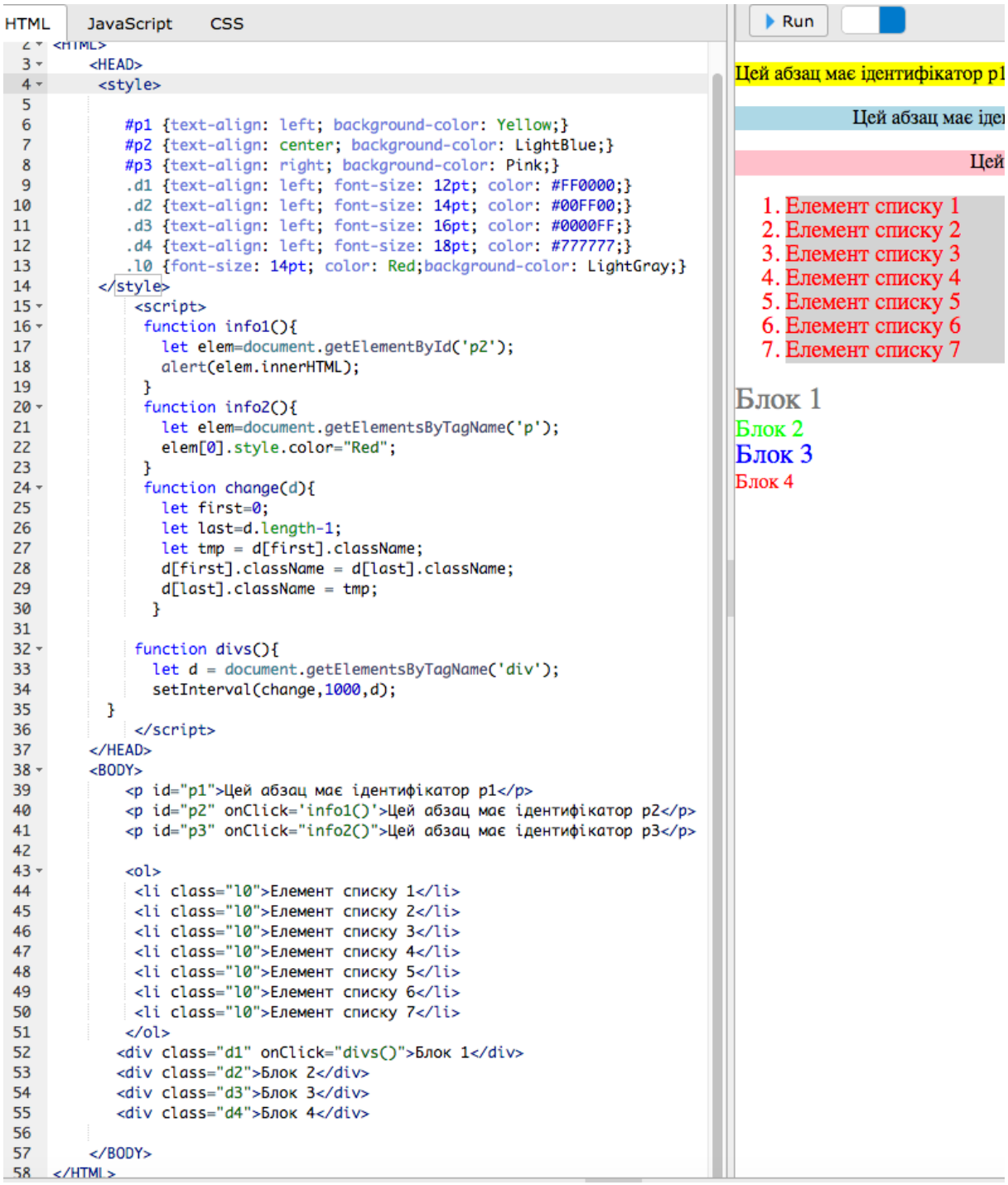


Рис. 9.2.4. Виконання Завдання 4

Завдання 5. Написати код на JavaScript, за яким буде реалізовано горизонтальний рух блоку div вперед-назад (блок переміщується на 300 пікселів праворуч, потім на 300 пікселів ліворуч і т.д.) Рух зупиняється при натисненні лівої кнопки миші на блоці.

Код вебсторінки, відповідної розв'язку, може бути таким (див. Рис. 9.2.5):

```
<!DOCTYPE html>
<HTML>
<HEAD>
 <style>
```

```
 div{
 font-size: 20pt;
 margin:20px;
 color:green;
 position:absolute;
 left: 20px;
 }
 </style>
 </HEAD>
<BODY>
 <div onClick="stop()">Рухомий блок</div>

 <script>
let d = document.getElementsByTagName('div')[0];
let direction = 'right';
let distance = 0;

function line() {
 if (direction == 'right'){
 distance++;
 if (distance == 300) direction = 'left';
 }
 if (direction == 'left'){
 distance--;
 if (distance == 0) direction = 'right';
 }

d.style.left = distance + 'px';
}

let move = setInterval(line,10);

function stop(){
clearInterval(move);
}
 </script>
</BODY>
</HTML>
```

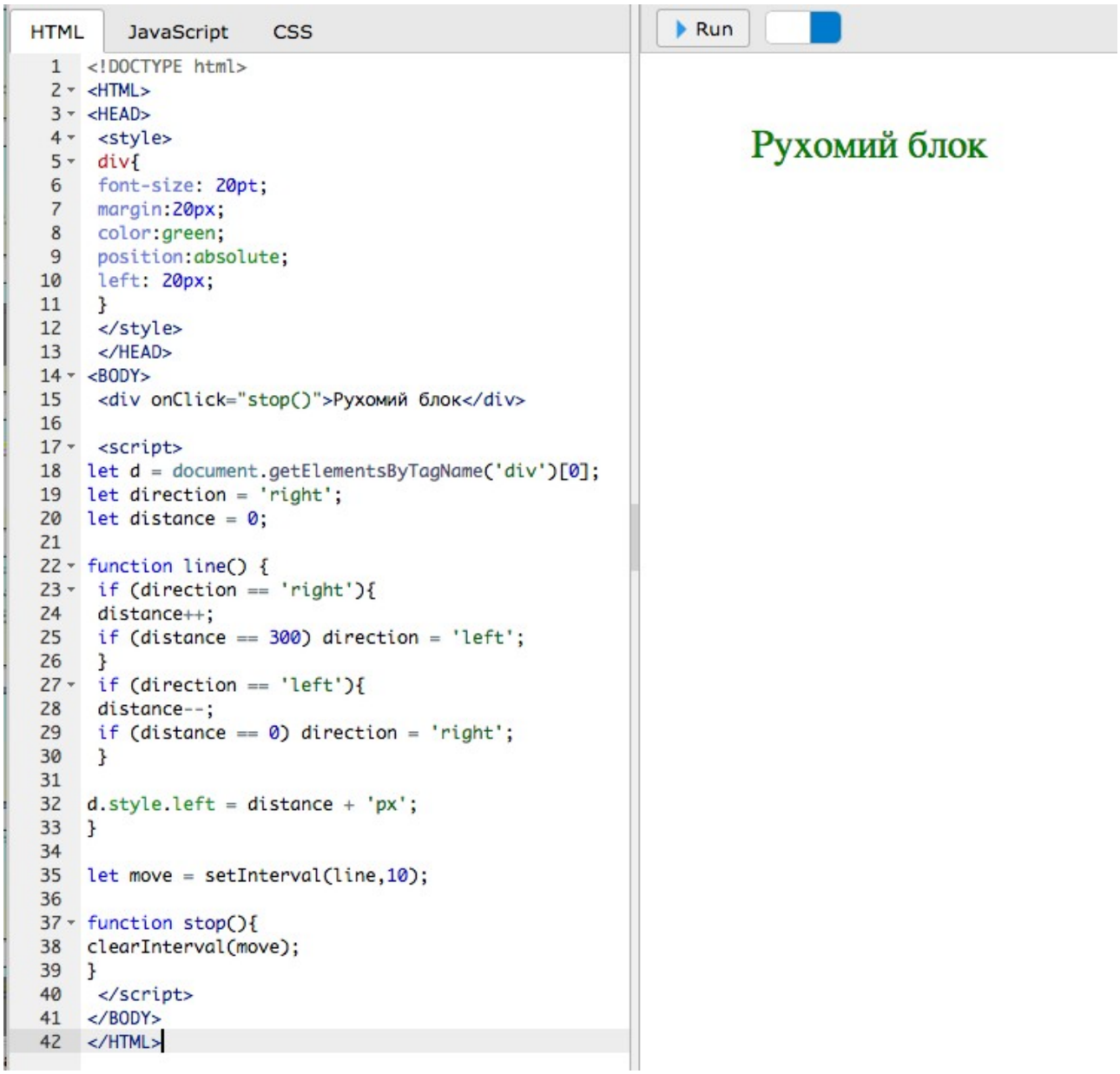


Рис. 9.2.5. Виконання Завдання 5

Практична робота №3
Створення форми в HTML-документі

Завдання 1. Створити HTML-документ з формою за зразком (Рис. 9.3.1):

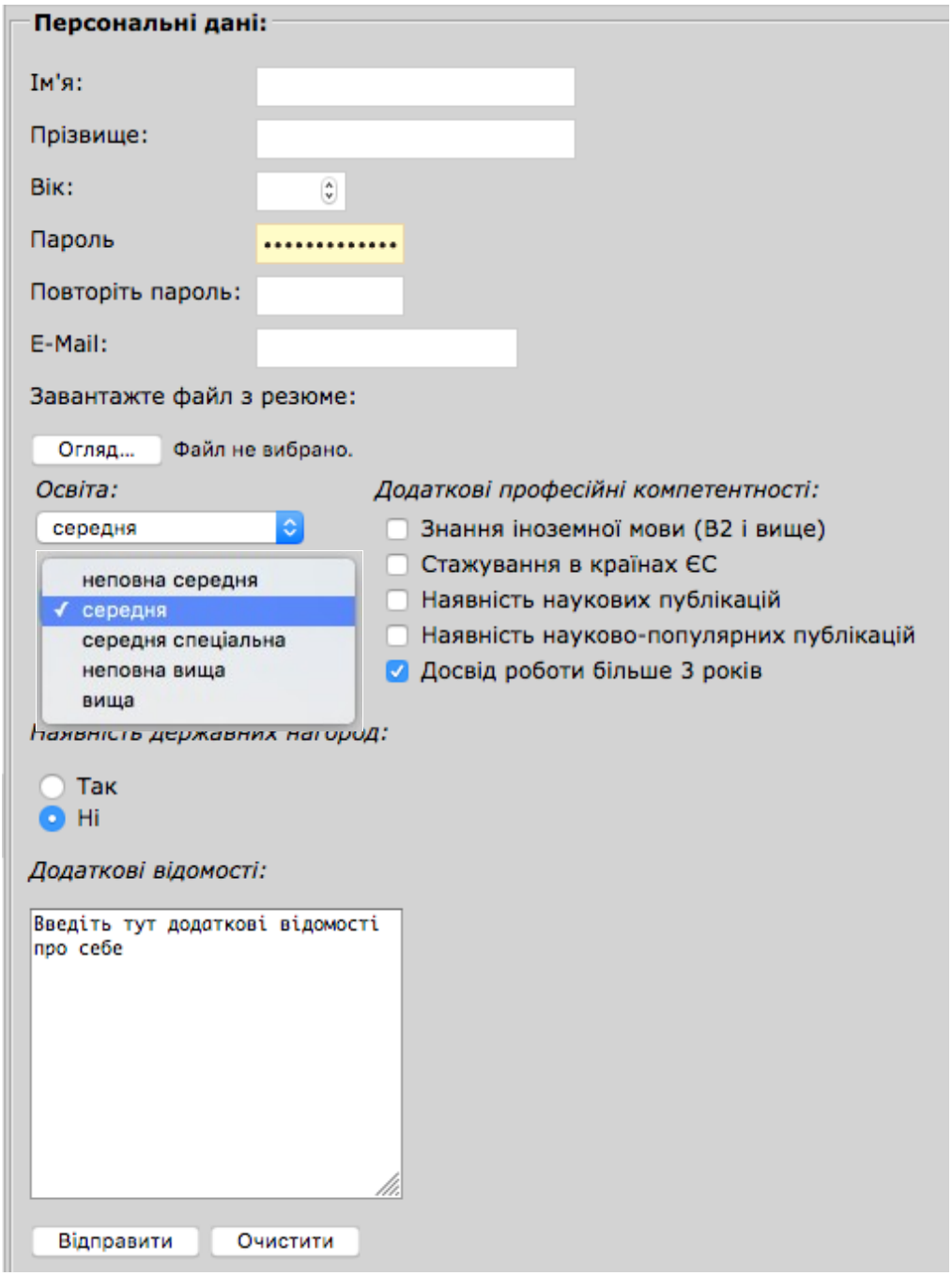


Рис. 9.3.1. Зразок форми.

На рисунку список "Освіта" зображений і у згорнутому, і в розгорнутому вигляді.

Встановити такі обмеження для даних, що вводяться у форму:

поля Ім'я, Прізвище, Вік є обов'язковими для введення;
допустимі межі для значення у полі Вік- від 0 до 120;
мінімальна довжина пароля - 4 символи.

Код вебсторінки, відповідної розв’язку, може бути таким (див. Рис. 9.3.2):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
        body {font-family: Verdana,Tahoma; font-size:12px;}
        form {background-color: LightGray;}
        .small {font-size:10px;}
        .inp {font-size:11px;position:absolute;left:22%;}
    </style>
  </head>
  <body>
    <form>
      <fieldset>
        <legend><b>Персональні дані:</b></legend>
         <p>Ім'я: <input class="inp" type="text" name="first_name"
size="25" required></p>
                  <p>Прізвище:  <input  class="inp"  type="text"
name="last_name" size="25" required ></p>
             <p>Вік: <input class="inp" type="number" name="age"
size="5" min="0" max="120" required ></p>
         <p>Пароль: <input class="inp" type="password" name="pass1"
size="10" minlength="4"></p>
          <p>Повторіть пароль: <input class="inp" type="password"
name="pass2" size="10" minlength="4"></p>
                  <p>E-Mail:  <input  class="inp"  type="e-mail"
name="mail"></p>
        <p>Завантажте файл з резюме:</p>
        <input type="file" name="resume">

       <table>
        <tr>
                                          <td><i>Освіта:</i></td><td
width="7%"></td><td><i>Додаткові                          професійні
компетентності:</i></td>
        </tr>
        <tr>
        <td valign="top">
        <select>
          <option value="unsecondary">неповна середня</option>
          <option value="secondary" selected>середня</option>
          <option value="special">середня спеціальна</option>
          <option value="unhigher">неповна вища</option>
          <option value="higher">вища</option>
        </select>
        </td>
        <td></td>
```

```
        <td valign="top">
          <input type="checkbox" name="option1" value="p1"> Знання
іноземної мови (B2 і вище)<br>
               <input type="checkbox" name="option2" value="p2">
Стажування в країнах ЄС <br>
               <input type="checkbox" name="option3" value="p3">
Наявність наукових публікацій<br>
               <input type="checkbox" name="option4" value="p4">
Наявність науково-популярних публікацій<br>
                <input type="checkbox" name="option1" value="p5"
checked> Досвід роботи більше 3 років<br>
        </td>
        </tr>
       </table>

      <p><i>Наявність державних нагород:</i></p>
        <input name="award" type="radio" value="yes"> Так <br>
         <input name="award" type="radio" value="no" checked > Ні
<br>

      <p><i>Додаткові відомості:</i></p>
              <p><textarea  class="small"  name="add"  rows="10"
cols="30">Введіть тут додаткові відомості про себе</textarea></p>

     <input type="submit" name="submitInfo" value="Відправити">
     <input type="reset"  name="ResetInfo" value="Очистити">
      </fieldset>
    </form>
  </body>
</html>
```

```
<title>Заголовок</title>
<style>
    body {font-family: Verdana,Tahoma; font-size:12px;}
    form {background-color: LightGray;}
    .small {font-size:10px;}
    .inp {font-size:11px;position:absolute;left:22%;}
</style>
</head>
<body>
<form>
  <fieldset>
    <legend><b>Персональні дані:</b></legend>
    <p>Ім'я: <input class="inp" type="text" name="first_name" size="25" required></p>
    <p>Прізвище: <input class="inp" type="text" name="last_name" size="25" required ></p>
    <p>Вік: <input class="inp" type="number" name="age" size="5" min="0" max="120" required ></p>
    <p>Пароль: <input class="inp" type="password" name="pass1" size="10" minlength="4"></p>
    <p>Повторіть пароль: <input class="inp" type="password" name="pass2" size="10" minlength="4"></p>
    <p>E-Mail: <input class="inp" type="e-mail" name="mail"></p>
    <p>Завантажте файл з резюме:</p>
    <input type="file" name="resume">

   <table>
    <tr>
    <td><i>Освіта:</i></td><td width="7%"></td><td><i>Додаткові професійні компетентності:</i></td>
    </tr>
    <tr>
    <td valign="top">
    <select>
      <option value="unsecondary">неповна середня</option>
      <option value="secondary" selected>середня</option>
      <option value="special">середня спеціальна</option>
      <option value="unhigher">неповна вища</option>
      <option value="higher">вища</option>
    </select>
    </td>
    <td></td>
    <td valign="top">
      <input type="checkbox" name="option1" value="p1"> Знання іноземної мови (B2 і вище)<br>
      <input type="checkbox" name="option2" value="p2"> Стажування в країнах ЄС <br>
      <input type="checkbox" name="option3" value="p3"> Наявність наукових публікацій<br>
      <input type="checkbox" name="option4" value="p4"> Наявність науково-популярних публікацій<br>
      <input type="checkbox" name="option1" value="p5" checked> Досвід роботи більше 3 років<br>
    </td>
    </tr>
   </table>
  <p><i>Наявність державних нагород:</i></p>
    <input name="award" type="radio" value="yes"> Так <br>
    <input name="award" type="radio" value="no" checked > Ні <br>
  <p><i>Додаткові відомості:</i></p>
  <p><textarea class="small" name="add" rows="10" cols="30">Введіть тут додаткові відомості про себе</textarea></p>
 <input type="submit" name="submitInfo" value="Відправити">
 <input type="reset"  name="ResetInfo" value="Очистити">
</fieldset>
</form>
</body>
</html>
```

Рис. 9.3.1. Виконання Завдання 1

Практична робота №4

Робота з формою в HTML-документі

Створити HTML-документ з формою за зразком (див. Практичну роботу №3, Рис.9.4.1):

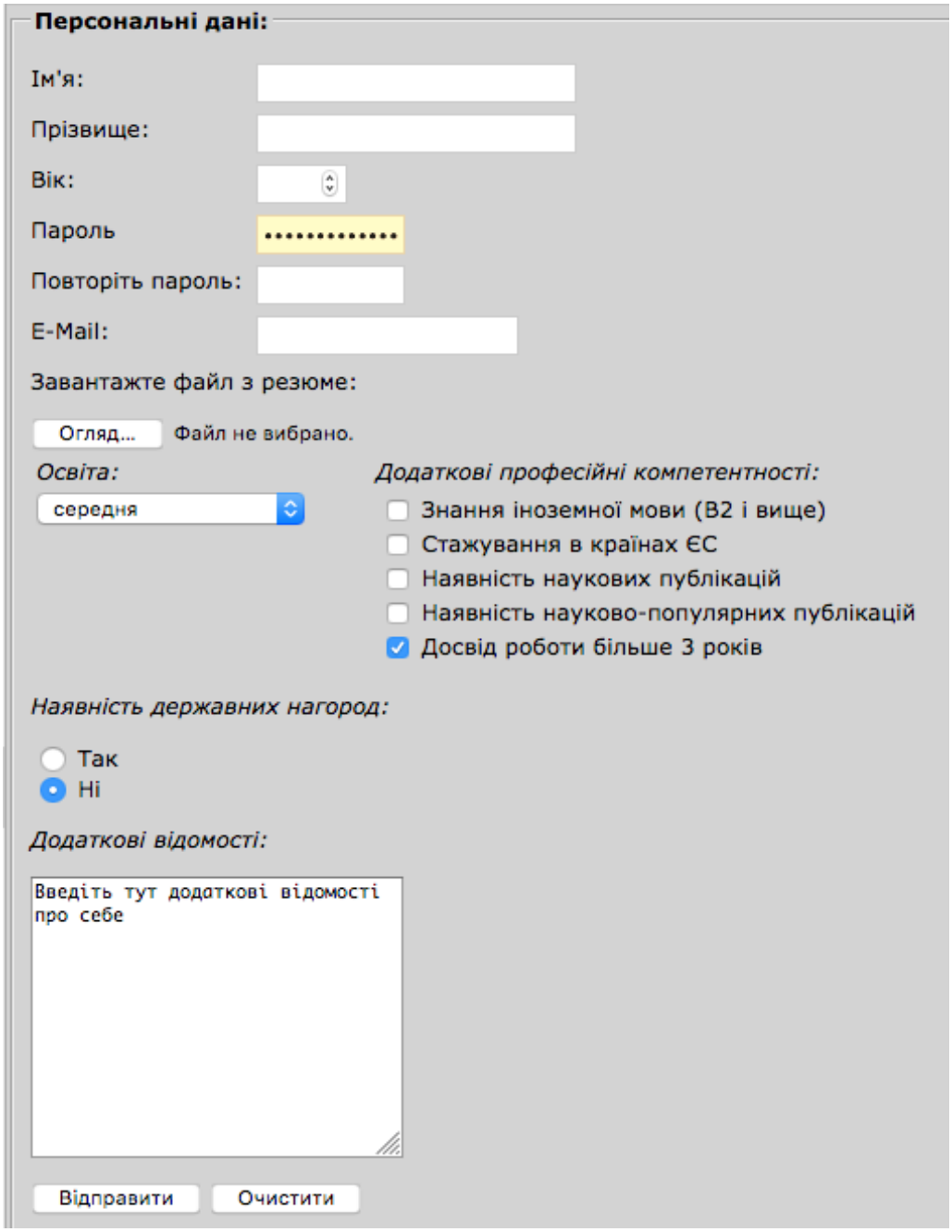


Рис. 9.4.1. Зразок форми.

Використовуючи засоби JavaScript, забезпечити виконання таких дій з формою:

Завдання 1. Забезпечити перевірку введеного імені (поле Ім'я) після виходу з поля:

якщо введене користувачем ім’я введено вперше / було використане раніше (перевіряється попередньо створений масив імен), користувачеві видається повідомлення виду: фоновий колір поля - зелений / червоний.

Завдання 2. Забезпечити перевірку введеного прізвища (поле Прізвище) після виходу з поля:

якщо введене користувачем прізвище введено вперше / було використане раніше (перевіряється попередньо створений масив прізвищ), користувачеві видається повідомлення виду: текст поряд з полем “Допустиме” (зеленого кольору) або “Недопустиме” (червоного).

Завдання 3. Забезпечити перевірку введеного віку (поле Вік) після виходу з поля:

якщо введений користувачем вік не менше 18 і не більше 65, користувачеві видається повідомлення виду: зображення поряд з полем зі знаком +, інакше - зі знаком - . Зображення створити самостійно.

Завдання 4. Якщо користувачем вказана наявність державних нагород (вибрана радіокнопка Так), встановити світло-зелений фон форми, інакше - світло-жовтий.

Завдання 5. Забезпечити перевірку даних, введених у форму, та відповідне інформування користувача при натисненні кнопки Відправити.

Має перевірятися:

коректне заповнення полів Ім’я та Прізвище,

збіг даних, введених у поля Пароль та Повторіть пароль,

наявність користувацького вмісту в полі Додаткові відомості.

Для інформування користувачеві мають бути виведені вікна з привітанням та вказанням імені, прізвища, додаткових відомостей. У випадку виявлення розбіжності у полях введення пароля та відсутності додаткових відомостей мають бути виведені відповідні повідомлення.

Завдання 6. Забезпечити повернення всіх полів до початкового стану при натисненні кнопки Очистити.

Код вебсторінки, відповідної розв’язку, може бути таким (див. Рис. 9.4.2, 9.4.3):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
        body {font-family: Verdana,Tahoma; font-size:12px;}
        form {background-color: LightGray;}
```

```
        .small {font-size:10px;}
        .inp {font-size:11px;position:absolute;left:22%;}
        #check {font-size:11px;position:absolute;left:50%;}
        #ag {font-size:11px;position:absolute;left:35%;}
    </style>
  </head>
  <body>
    <form name=f>
      <fieldset>
        <legend><b>Персональні дані:</b></legend>
         <p>Ім'я: <input class="inp" type="text" name="first_name"
size="25" onBlur="check_fn()"></p>
                    <p>Прізвище:  <input  class="inp"  type="text"
name="last_name"        size="25"        onBlur="check_ln()"><span
id="check"></span></p>
             <p>Вік: <input class="inp" type="number" name="age"
size="5" value="18" onBlur="check_a()">
             <input type="image" name="img"   alt="0" width="18"
height="18" id="ag" hidden></p>
          <p>Пароль: <input class="inp" name="p1" type="password"
id="p1" size="10"></p>
              <p>Повторіть пароль: <input class="inp" name="p2"
type="password" id="p2" size="10"></p>
              <p>E-Mail: <input class="inp" <input type="email"
name="email" placeholder='name@domain.com'></p>
        <p>Завантажте файл з резюме:</p>
        <input type="file" name="resume">
       <table>
        <tr>
                                          <td><i>Освіта:</i></td><td
width="7%"></td><td><i>Додаткові                          професійні
компетентності:</i></td>
        </tr>
        <tr>
        <td valign="top">
        <select>
          <option value="unsecondary">неповна середня</option>
          <option value="secondary" selected>середня</option>
          <option value="special">середня спеціальна</option>
          <option value="unhigher">неповна вища</option>
          <option value="higher">вища</option>
        </select>
        </td>
        <td></td>
        <td valign="top">
          <input type="checkbox" name="option1" value="p1"> Знання
іноземної мови (B2 і вище)<br>
               <input  type="checkbox"  name="option2"  value="p2">
Стажування в країнах ЄС <br>
               <input  type="checkbox"  name="option3"  value="p3">
Наявність наукових публікацій<br>
               <input  type="checkbox"  name="option4"  value="p4">
Наявність науково-популярних публікацій<br>
```

```
                <input type="checkbox" name="option5" value="p5"
checked> Досвід роботи більше 3 років<br>
        </td>
        </tr>
       </table>

      <p><i>Наявність державних нагород:</i></p>
        <input name="award" type="radio" id="yes"> Так <br>
        <input name="award" type="radio" id="no" checked> Ні <br>

      <p><i>Додаткові відомості:</i></p>
        <p><textarea class="small" name="add" rows="10" cols="30"
id="area"></textarea></p>

       <button type="button" name="SubmitInfo" value="Відправити"
onclick="send()">Відправити</button>
     <input type="reset"  name="ResetInfo" value="Очистити">
      </fieldset>
    </form>
    <script>

        let fn=f.first_name;
        let arr_fn=[];
        let nm='';

        function check_fn() {
          let p = true;
          if (fn.value != ''){
          for (let i = 0; i < arr_fn.length; i++){
            if (arr_fn[i] == fn.value){
               fn.style.background = "red";
               p = false;
               nm='minus';
            break;
          }
          }
         if (p){
           fn.style.background = "green";
           arr_fn.push(fn.value);
           nm='plus';
           }
           }
        }

        let ln=f.last_name;
        let arr_ln=[];
        let ch=document.getElementById("check");
        function check_ln() {
          let p = true;
          if (ln.value != ''){
          for (let i = 0; i < arr_ln.length; i++){
            if (arr_ln[i] == ln.value){
               ch.style.color = "red";
```

```
        ch.textContent = "Недопустиме";
        p = false;
     break;
   }
   }
 if (p){
    ch.style.color = "green";
    ch.textContent = "Допустиме";
    arr_ln.push(ln.value);
    }
    }
}

let a=f.age;
let img1 = document.getElementById("ag");

function check_a() {
  if (a.value >= 18 && a.value<=65) {
   img1.src="plus.png"
   img1.hidden=false;
  }
  else {
   img1.src="minus.png"
   img1.hidden=false;
  }
 }

let y=document.getElementById("yes");
let n=document.getElementById("no");

y.onclick = function() {

 f.style.backgroundColor='lightgreen';

}

n.onclick = function() {

 f.style.backgroundColor="lightyellow";

}


 function send() {

 if (ch.textContent = "Допустиме") {
 surname=ln.value;
 }
 if (nm='plus') {
 name=fn.value;
 }

 alert('Вітаю, '+name+' '+surname+'!');
```

```
        let x1 = document.getElementById("p1").value;
        let x2 = document.getElementById("p2").value;

        if (x1!=x2) {
        alert('Перевірте збіг паролів!')
        }

        let ar=document.getElementById("area").value;
        if (ar=='') {
        alert('Ви не ввели додаткові відомості про себе!');

        }
        else {
        m=f.add.value;
        }
        alert('Додаткові відомості про Вас: '+m);
        }

        </script>
  </body>
</html>
```

Зразок заповненої форми:

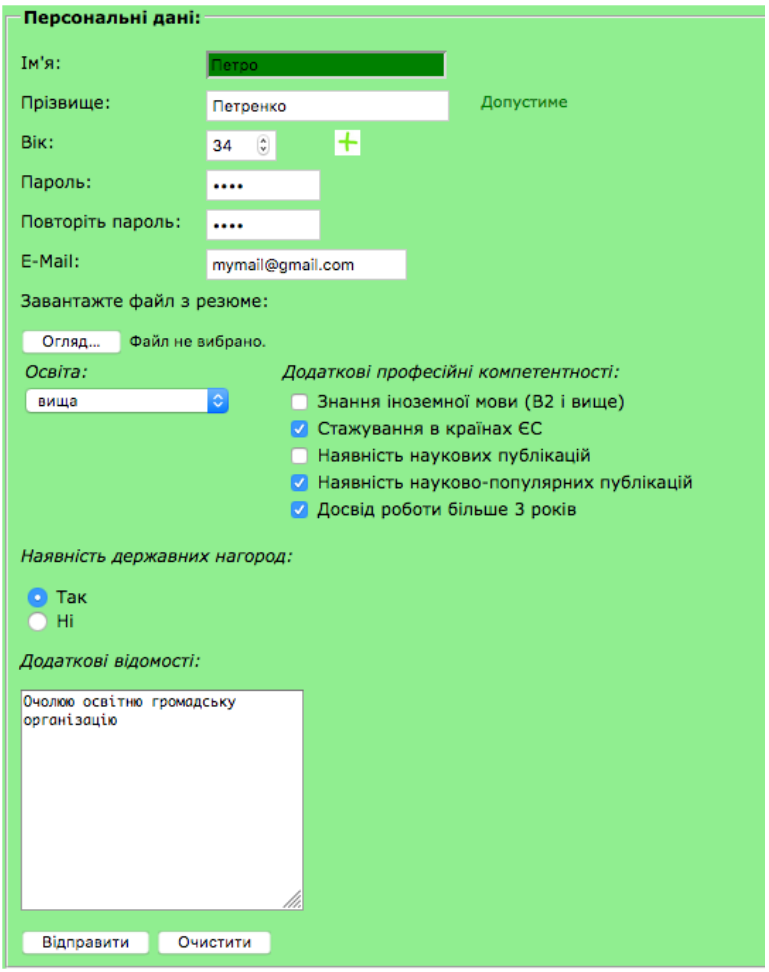


Рис. 9.4.2. Зразок заповненої форми.

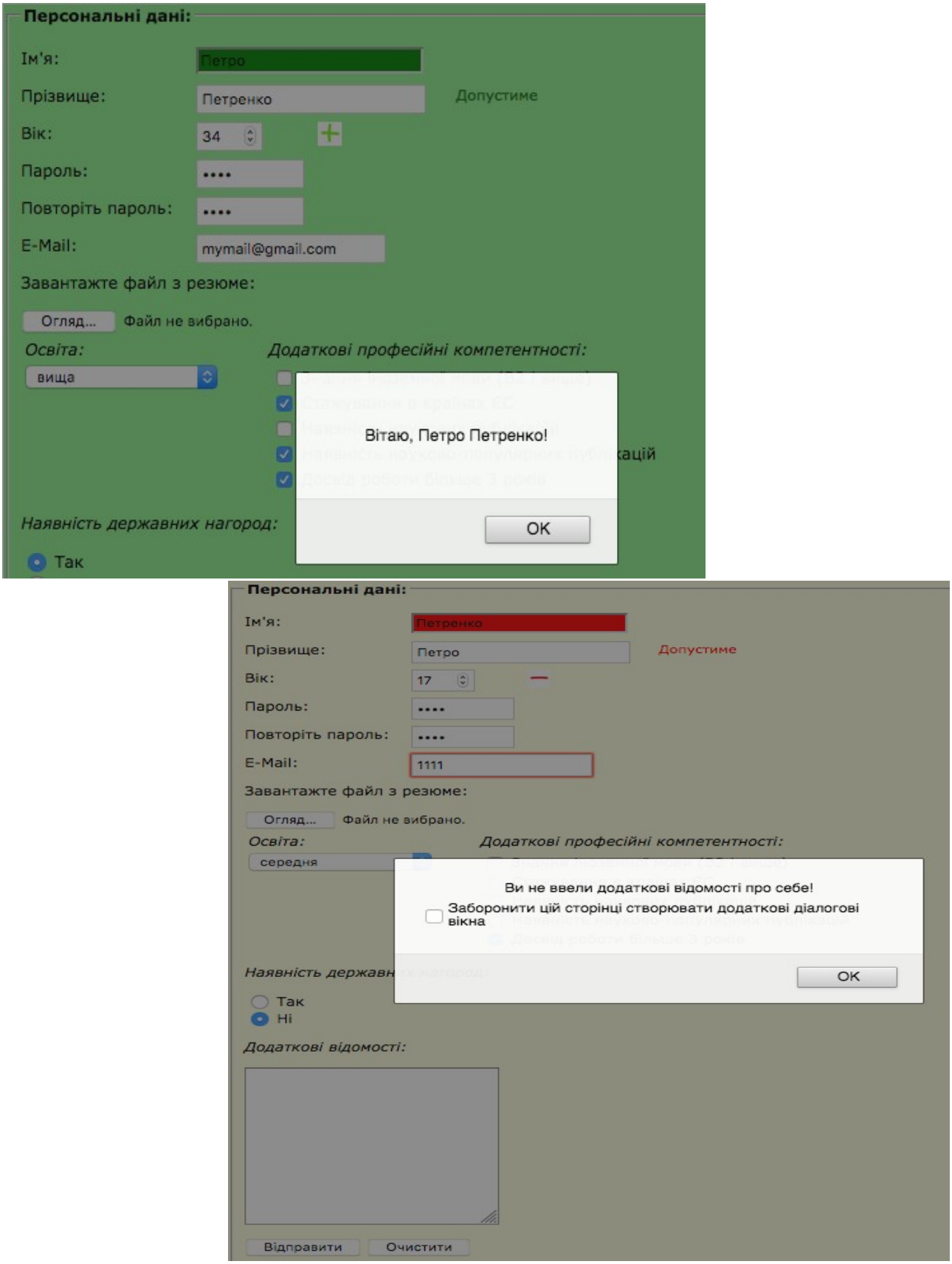


Рис. 9.4.3. Виконання Завдання 5.

Практична робота №5
Внесення змін в HTML-документ

Завдання 1. Створити html-документ з формою виду (Рис. 9.5.1):

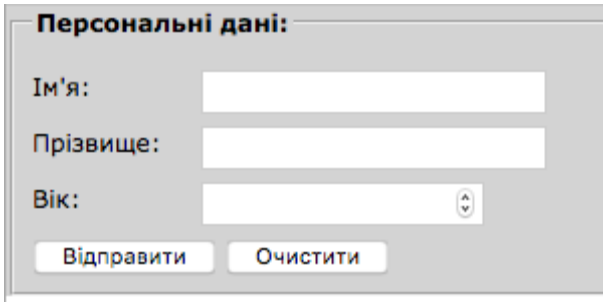


Рис. 9.5.1. Приклад форми.

Коректне заповнення полів Ім’я та Прізвище передбачає введення в них непорожніх значень.

Коректне заповнення поля Вік передбачає введення в нього значення, що не менше 18.

У випадку коректного заповнення всіх полів при натисненні кнопки Відправити мають бути виконані такі дії:

- виводиться вікно з привітанням, в якому міститься ім’я і прізвище користувача;
- на сторінці під формою виводиться повідомлення “Користувач пройшов перевірку”;
- нижче виводиться нумерований список, елементами якого є: ім’я, прізвище, вік користувача;
- нижче виводиться випадний список, елементами якого є ім’я, прізвище, вік користувача, вибраним елементом списку є вік.

У випадку некоректного заповнення полів при натисненні кнопки Відправити має виводитись повідомлення “Неправильні дані”.

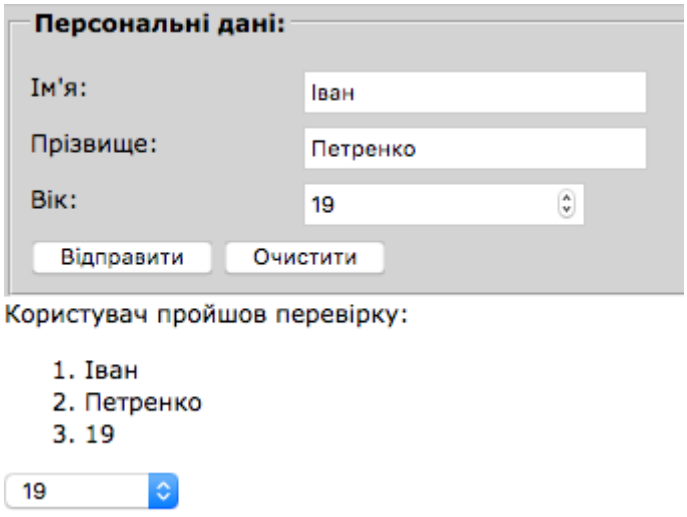


Рис. 9.5.2. Приклад фрагменту сторінки з результатом виконання програми.

Код вебсторінки, відповідної розв’язку, може бути таким:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
        body {font-family: Verdana,Tahoma; font-size:12px;}
        form {background-color: LightGray;}
        .inp {font-size:11px;position:absolute;left:22%;}
    </style>
  </head>
  <body>
    <form name=f>
      <fieldset>
        <legend><b>Персональні дані:</b></legend>
         <p>Ім'я: <input class="inp" type="text" name="first_name"
size="25"></p>
                    <p>Прізвище:  <input  class="inp"  type="text"
name="last_name" size="25"></p>
        <p>Вік: <input class="inp" type="number" name="age">
        </p>

       <button type="button" name="SubmitInfo" value="Відправити"
onclick="send()">Відправити</button>
     <input type="reset"  name="ResetInfo" value="Очистити">
      </fieldset>
    </form>

<ol id="ol">
</ol>

<select id="sel" hidden >
</select>
   <script>
    function send() {

     let fn=f.first_name;
     let ln=f.last_name;
     let a=f.age;

         if (fn.value !='' && ln.value !='' && a.value >=18) {
          alert('Вітаю, '+fn.value+' '+ln.value+'!');

          ol.before('Користувач пройшов перевірку:');

          let li1 = document.createElement('li');
          li1.innerHTML = fn.value;
          ol.append(li1);
          let li2 = document.createElement('li');
          li2.innerHTML = ln.value;
          ol.append(li2);
          let li3 = document.createElement('li');
          li3.innerHTML = a.value;
```

```
          ol.append(li3);

          let opt1 = document.createElement('option');
          opt1.innerHTML = fn.value;
          sel.append(opt1);
          let opt2 = document.createElement('option');
          opt2.innerHTML = ln.value;
          sel.append(opt2);
          let opt3 = document.createElement('option');
          opt3.innerHTML = a.value;
          opt3.selected=true;
          sel.append(opt3);

          sel.hidden = false;
         }
         else {
          alert('Неправильні дані!');
         }
     }
   </script>
  </body>
</html>
```

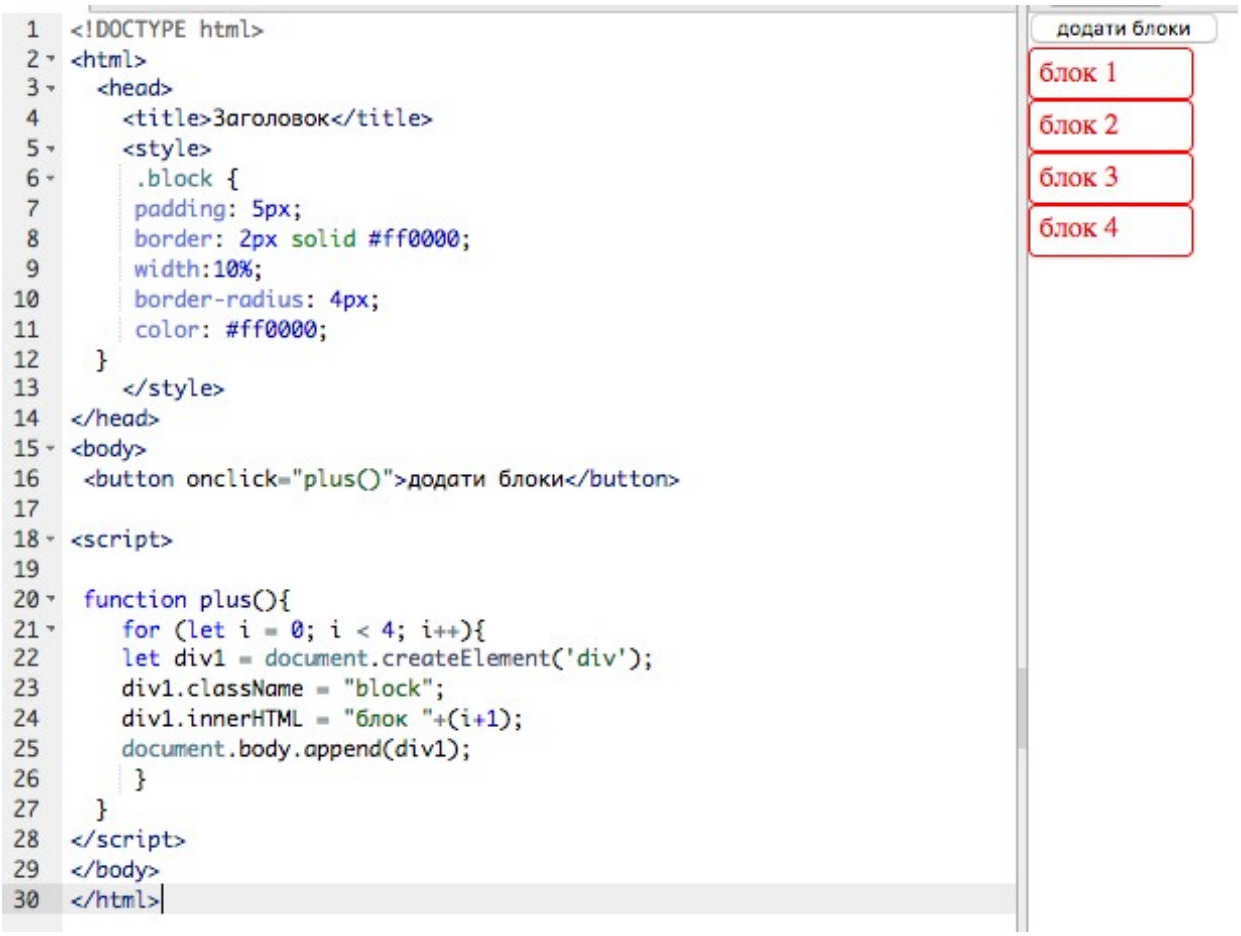
Завдання 2. Створити документ з кнопкою на сторінці: «Додати блоки». За натисненням на кнопку додаються 4 блоки div з текстами “блок 1”, ..., “блок 4”. Код вебсторінки, відповідної розв’язку, може бути таким (див. Рис. 9.5.3):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
     .block {
     padding: 5px;
     border: 2px solid #ff0000;
     width:10%;
     border-radius: 4px;
     color: #ff0000;
  }
    </style>
</head>
<body>
 <button onclick="plus()">додати блоки</button>

<script>

 function plus(){
     for (let i = 0; i < 4; i++){
    let div1 = document.createElement('div');
    div1.className = "block";
    div1.innerHTML = "блок "+(i+1);
    document.body.append(div1);
```

```
    }
  }
</script>
</body>
</html>
```




```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
     .block {
     padding: 5px;
     border: 2px solid #ff0000;
     width:10%;
     border-radius: 4px;
     color: #ff0000;
   }
    </style>
</head>
<body>
 <button onclick="plus()">додати блоки</button>

<script>

 function plus(){
    for (let i = 0; i < 4; i++){
    let div1 = document.createElement('div');
    div1.className = "block";
    div1.innerHTML = "блок "+(i+1);
    document.body.append(div1);
     }
  }
</script>
</body>
</html>
```

Рис. 9.5.3. Виконання Завдання 2.

Завдання 3. Доповнити сторінку, створену у Завданні 2: додати на сторінку кнопку “Видалити блоки”. При натисненні на кнопку запитується номер блоку для видалення. Якщо такий блок існує, він видаляється, інакше виводиться вікно з повідомленням про помилку.

Код вебсторінки, відповідної розв’язку, може бути таким (див. Рис. 9.5.4):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
     .block {
     padding: 5px;
     border: 2px solid #ff0000;
     width:10%;
     border-radius: 4px;
```

```
     color: #ff0000;
  }
    </style>
</head>
<body>
 <button onclick="plus()">додати блоки</button>
 <button onclick="minus()">видалити блоки</button>
<script>

 function plus(){
    for (let i = 0; i < 4; i++){
    let div1 = document.createElement('div');
    div1.className = "block";
    div1.innerHTML = "блок "+(i+1);
    document.body.append(div1);
     }
  }
 function minus(){
    let n = prompt('Видалити блок № (1-4):');
    if ( n >= 1  &&  n <= 4 ) {
    let del = document.getElementsByTagName('div')[n - 1];
    del.remove();
    }
    else {
     alert('Неправильний номер блоку!');
    }
}

</script>
</body>
</html>
```

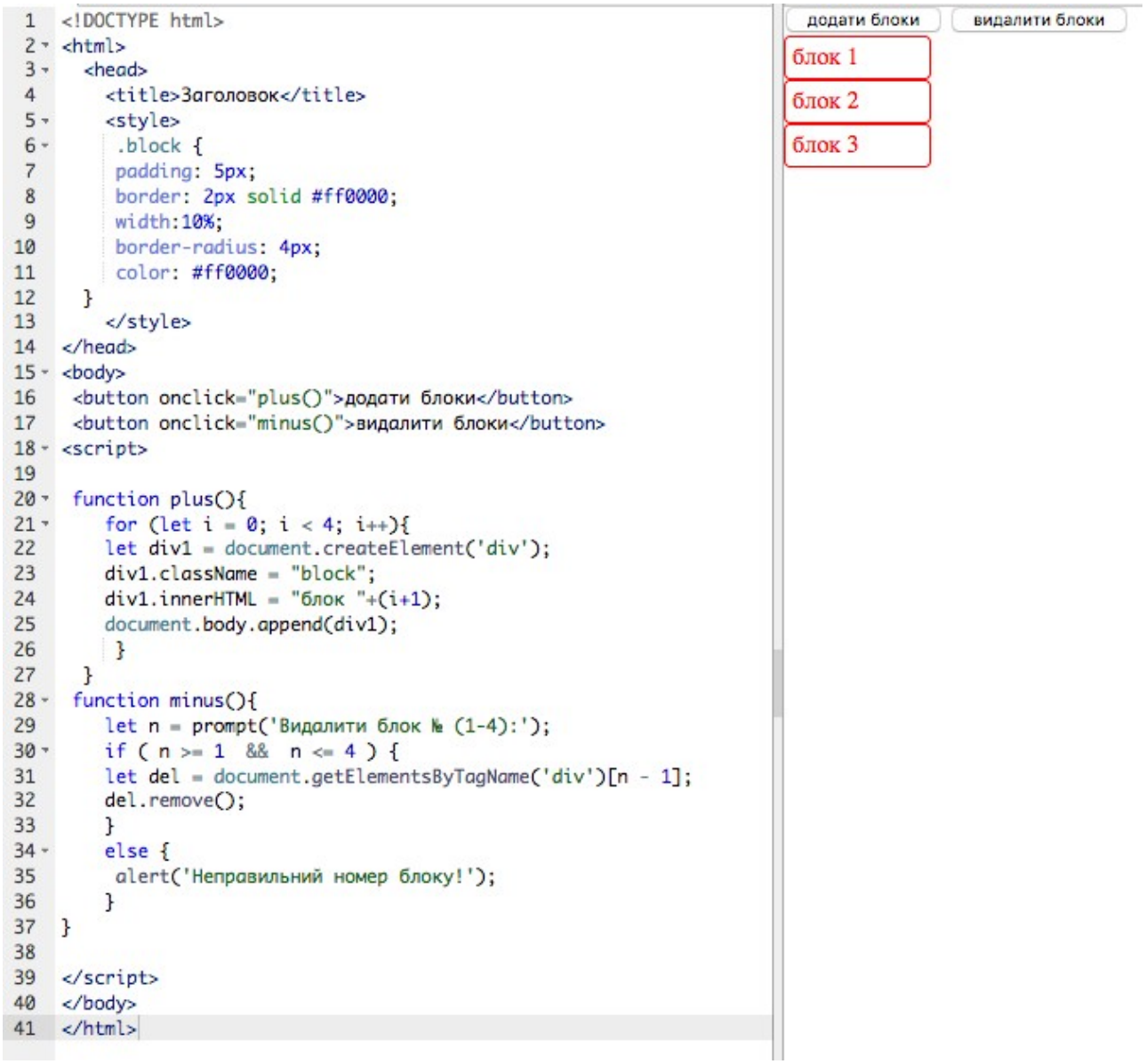


Рис. 9.5.4. Приклад результату виконання Завдання 3 після видалення блоку 4

Завдання 4. Створити документ з кнопкою на сторінці: «Додати блоки». За натисненням на кнопку додаються 4 блоки div з текстами “Блок 1”, ..., “Блок 4”. Інформація про блоки має зберігатися в масиві об’єктів, окремий об’єкт має структуру:

```
{     title: "текст", // Блок №
      width: "50px",
      height: "50px",
 };
```

Код вебсторінки, відповідної розв’язку, може бути таким (див. Рис. 9.5.5):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
     .block {
     padding: 5px;
```

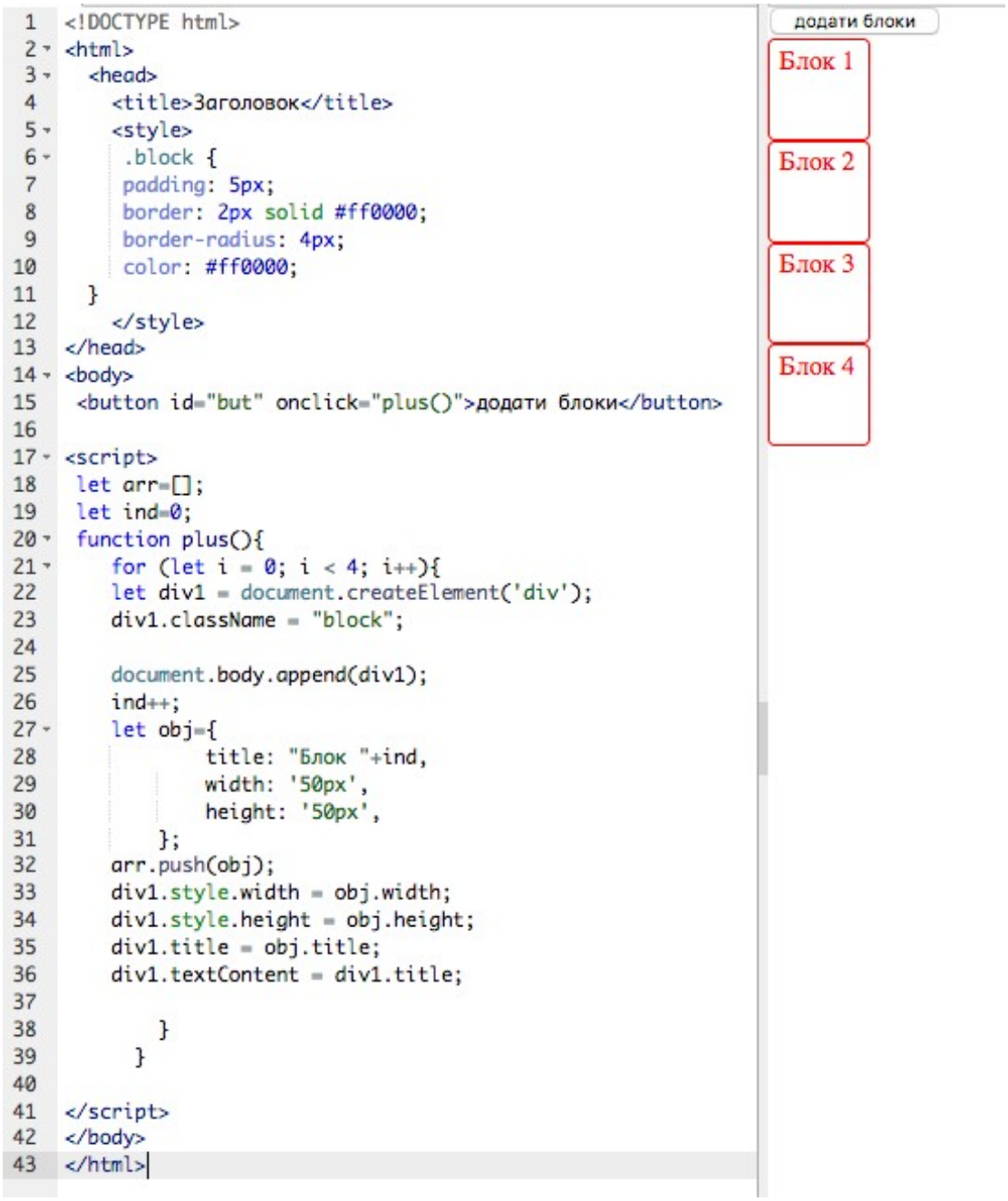
```
     border: 2px solid #ff0000;
     border-radius: 4px;
     color: #ff0000;
  }
    </style>
</head>
<body>
 <button onclick="plus()">додати блоки</button>

<script>
 let arr=[];
 let ind=0;
 function plus(){
     for (let i = 0; i < 4; i++){
    let div1 = document.createElement('div');
    div1.className = "block";

    document.body.append(div1);
    ind++;
    let obj={
              title: "Блок "+ind,
               width: '50px',
               height: '50px',
          };
     arr.push(obj);
     div1.style.width = obj.width;
     div1.style.height = obj.height;
     div1.title = obj.title;
     div1.textContent = div1.title;

         }
          }

</script>
</body>
</html>
```




```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
     .block {
     padding: 5px;
     border: 2px solid #ff0000;
     border-radius: 4px;
     color: #ff0000;
   }
    </style>
</head>
<body>
 <button id="but" onclick="plus()">додати блоки</button>

<script>
 let arr=[];
 let ind=0;
 function plus(){
    for (let i = 0; i < 4; i++){
    let div1 = document.createElement('div');
    div1.className = "block";

    document.body.append(div1);
    ind++;
    let obj={
            title: "Блок "+ind,
            width: '50px',
            height: '50px',
        };
    arr.push(obj);
    div1.style.width = obj.width;
    div1.style.height = obj.height;
    div1.title = obj.title;
    div1.textContent = div1.title;

        }
      }

</script>
</body>
</html>
```

Рис. 9.5.5. Виконання Завдання 4.

Завдання 5. Доповнити сторінку, створену у Завданні 4: додати на сторінку кнопку “Додати тексти”. При натисненні на кнопку під кожним блоком (виведеним в результаті натиснення кнопки “Додати блоки”) має виводитись текст, значення якого збігається зі значенням атрибуту title відповідного блоку.

Код вебсторінки, відповідної розв’язку, може бути таким (див. Рис. 9.5.6):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Заголовок</title>
    <style>
     .block {
     padding: 5px;
     border: 2px solid #ff0000;
```

```
     border-radius: 4px;
     color: #ff0000;
  }
    </style>
</head>
<body>
 <button onclick="plus()">додати блоки</button>
 <button onclick="plus_title()">додати тексти</button>
<script>
 let arr=[];
 let ind=0;
 function plus(){
     for (let i = 0; i < 4; i++){
    let div1 = document.createElement('div');
    div1.className = "block";

    document.body.append(div1);
    ind++;
    let obj={
              title: "Блок "+ind,
               width: '50px',
               height: '50px',
          };
     arr.push(obj);
     div1.style.width = obj.width;
     div1.style.height = obj.height;
     div1.title = obj.title;
     div1.textContent = div1.title;
         }
       }
 function plus_title() {
    let divs = document.getElementsByTagName('div');
    for (let i = 0; i < divs.length; i++){
     divs[i].after(divs[i].title);
    }
 }
</script>
</body>
</html>
```

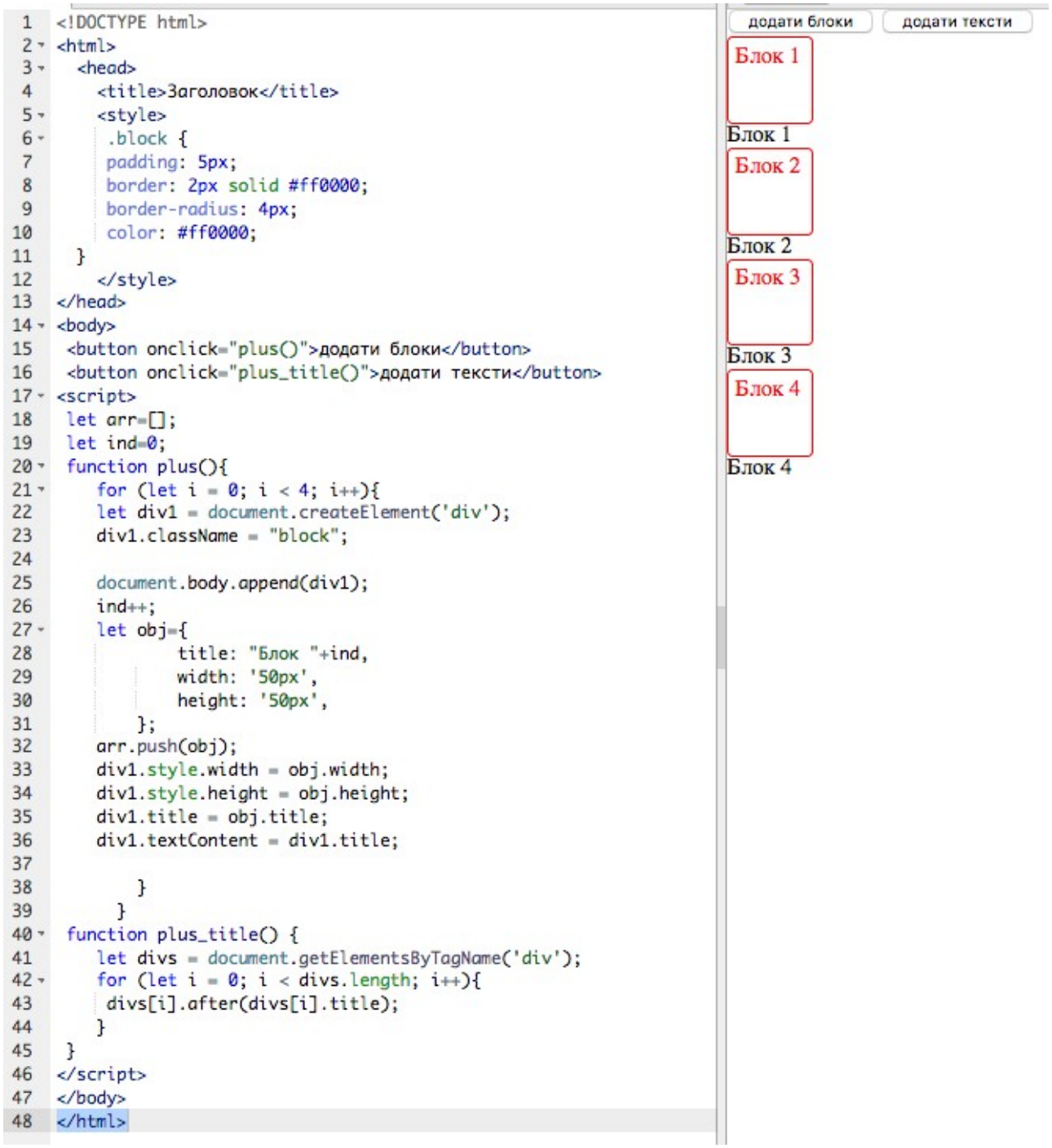


Рис. 9.5.6. Виконання завдання 5.

10. ЛАБОРАТОРНІ РОБОТИ
Лабораторна робота №1
JavaScript. Найпростіші сценарії

Завдання. Розробити дві HTML-сторінки, що містять скрипти JavaScript для розв'язування задач індивідуального варіанту. Скрипт для першої задачі має міститися у зовнішньому файлі. Скрипт для другої задачі має міститися в тексті HTML-сторінки. Передбачити введення і виведення даних у діалогових вікнах.

Для звіту з кожної задачі надіслати:

- HTML-документ (або його екранну копію),
- екранні копії, що демонструють введення вхідних даних і виведення результатів,
- файл сценарію .JS (або його екранну копію - для задачі 1)

Зауваження

Завдання виконувати в середовищі, яке вибрати самостійно.

*Кількість **повних** одиниць означає, що відповідь має бути цілим числом.*

Варіанти завдань для самостійного виконання

Варіант 1.

1. Задані сторони прямокутника a та b. Знайти його площу $S=a\cdot b$ та периметр $P=2\cdot(a+b)$.
2. Задана відстань L в кілометрах. Обчислити кількість повних морських миль в цій відстані (1 морська миля = 1,852 кілометри).

Варіант 2.

1. Задано радіус кола R. Знайти довжину кола $L=2\cdot\pi\cdot R$ та його площу $S=\pi\cdot R^2$. При обчисленні можна взяти $\pi\approx3{,}14$.
2. Задана маса M в кілограмах. Знайти кількість повних пудів в цій масі (1 пуд = 16,38 кілограмів).

Варіант 3.

1. Задано довжину ребра куба a. Знайти його площу поверхні $S=6\cdot a^2$ та об'єм $V=a^3$.
2. Навчальний тиждень студента повинен складати N занять. Скільки повних робочих днів повинно бути в навчальному тижні студента, якщо в один робочий день не може бути більше 3 занять?

Варіант 4.

1. Задано два додатних числа a та b. Знайти їх середнє арифметичне $(a+b)/2$ та середнє геометричне $\sqrt{a\cdot b}$.
2. Задана відстань L в метрах. Обчислити кількість повних кілометрів в цій відстані (1 кілометр = 1000 метрів).

Варіант 5.

1. Задані катети прямокутного трикутника a та b. Знайти його гіпотенузу $c=\sqrt{a^2+b^2}$ та периметр $P=a+b+c$.
2. Заданий розмір файлу N в байтах. Знайти кількість кластерів, що займає цей файл на зовнішньому магнітному носії (розмір одного кластеру – 4096 байт).

Варіант 6.

1. Задано довжину кола L. Знайти його радіус $R=L/(2\cdot\pi)$ та площу $S=\pi\cdot R^2$. При обчисленні можна взяти $\pi\approx 3{,}14$.
2. Студентські канікули складають N днів. Знайти кількість повних тижнів в студентських канікулах (1 тиждень = 7 днів).

Варіант 7.

1. Задано перший член арифметичної прогресії a та її різниця d. Знайти n-ний член прогресії $b=a+(n-1)\cdot d$ та її суму $S=n\cdot (a+b)/2$.
2. Задана відстань L в сантиметрах. Обчислити кількість повних дюймів в цій відстані (1 дюйм = 2,54 сантиметри).

Варіант 8.

1. Задана сторона a правильного трикутника. Знайти його площу $S=a^2\cdot\sqrt{3}/4$ та радіус вписаного кола $r=a\sqrt{3}/6$.
2. Тривалість навчального семестру складає T робочих днів. Знайти кількість повних робочих тижнів в навчальному семестрі (1 робочий тиждень = 5 робочих днів).

Варіант 9.

1. Визначити час падіння t тіла, вільно відпущеного з висоти h, та його швидкість V при ударі, якщо $h=g\cdot t^2/2$, $V=g\cdot t$. Опором повітря знехтувати, вважати величину вільного прискорення $g\approx 9{,}8$.
2. Заданий розмір файлу N в байтах. Знайти кількість повних кілобайт, що займає цей файл (1 кілобайт = 1024 байти).

Варіант 10.

1. Тіло рухається по колу радіуса R, здійснюючи повний оберт за час T. Обчислити швидкість руху тіла по колу $V=2\cdot\pi\cdot R/T$ та його доцентрове прискорення $a=V^2/R$.
2. Заданий період часу T в секундах. Знайти кількість повних хвилин в цьому періоді (1 хвилина = 60 секунд).

Варіант 11.

1. Тіло починає прямолінійний рівноприскорений рух з нерухомого стану із заданим прискоренням a. Знайти його швидкість $V=a \cdot t$ та пройдену відстань $S=a \cdot t^2/2$ через деякий заданий час t.
2. Заданий період часу T в годинах. Знайти кількість повних діб в цьому періоді (1 доба = 24 години).

Варіант 12.

1. Три опори R_1, R_2, R_3 з'єднані між собою. Знайти величину опорів послідовного з'єднання R_S ($R_S=R_1+R_2+R_3$) та паралельного з'єднання R_P ($1/R_P=1/R_1+1/R_2+1/R_3$).
2. Задана маса M в кілограмах. Знайти кількість повних тонн в цій масі (1 тонна = 1000 кілограмів).

Лабораторна робота №2
JavaScript. Перетворення типів

Завдання. Дати письмові відповіді.

Що буде виведено у результаті виконання інструкцій?

```
1. alert(1 / 0);

2. alert(1 / "not a number");

3. alert( NaN * 0 );

4. alert( 10 / NaN );

5. alert( 0 / 2 - 1 );

6. alert( "0" / 2 - 1 );

7. alert("2" +"0"+ 1 );

8. alert("2" -"0" - 1 );

9. alert(2 *("4" - 1) );

10. alert("2" + "4" - 1 );

11. alert("2" +"0" - 1 );

12. alert( 1 && undefined && null && "" && 0 );

13. alert("1" + "2" * "0");

14. alert( 1 && 0 && undefined && null && "" && "0" );

15. alert( 1 && null && "_" && 0 );

16. alert("1" * "2" * "0");

17. alert(3 + 2 + '1'+'0' );

18. alert( Boolean(1) );
```

1

```
19. alert( Boolean(0) );

20. alert(+"3"+ +"2" );

21. alert(+3+ +2 );

22. alert(2 % 2);

23. alert(false + true);

24. alert(false * true);

25. alert("група" > "трупа");

26. alert(true / false);

27. alert( 0 || null || "_" || 0 );

28. alert("резултат: ${2 + 2}");

29. alert( 0 || undefined || null || "" || false );

30. alert(`result: ${2 + 2}`);

31. let x = 2*2==4;
    alert(x);

32. let x;
    alert(x);

33. alert( 1 || 0 );

34. let name = "Student";
    alert( `hello ${"name"}` );

35. alert( Number(true) );
```

2

```
36. alert( 0 || undefined || null || "" || true );

37. alert( Boolean("")==Boolean(" "));

38. alert( Boolean("")==Boolean(0));

39. let a = -2;
    let b = a++;
    let c= ++a;
    alert(b);
    alert(c);

40. let a = 2;
    let b = 3;
    alert(!a + +b);

41. let n = 3;
    n *= 3;
    n += 3;
    alert( n );

42. let s = '2_' + '2';
    alert(s);

43. alert("5" > "10");

44. let a = -2;
    a++;
    ++a;
    alert( a );

45. let a = 1;
    let b = 2;
    alert(!a+String(b));
```

3

Лабораторна робота №3
JavaScript. Реалізація розгалужень

Завдання. Розробити HTML-сторінки, що містять скрипти JavaScript для розв'язування задач індивідуального варіанту. Кількість сторінок вибрати за бажанням. Спосіб використання скриптів вибрати за бажанням.
Передбачити введення і виведення даних у діалогових вікнах.
Для **звіту** з кожної задачі надіслати відповідні файли або екранні копії, що наочно відображають виконання завдань.
Зауваження:

- *Завдання виконувати в середовищі, яке вибрати самостійно.*
- *У першій задачі не передбачено використання функцій користувача (вхідні дані: змінні x і y, результат - змінна f). Але за бажання виконавець роботи може написати і використати власну функцію.*
- *У другій задачі передбачені текстові відповіді.*

Варіанти завдань для самостійного виконання
Варіант 1.

1. Обчислити значення функції $f(x,y)=\begin{cases} x^2+2y+1, якщо x>0, \\ \cos(x\cdot y), якщо x\le 0. \end{cases}$
2. Відома висота *h* на яку потрібно розмістити тіло відносно поверхні моря. Визначити де саме потрібно розмістити тіло відносно поверхні моря:

- підняти над поверхнею;
- розташувати на поверхні;
- занурити під поверхнею.

Варіант 2.

1. Обчислити значення функції $f(x,y)=\begin{cases} \sin(x)+y, якщо y<0, \\ x-y^2, якщо y\ge 0. \end{cases}$
2. Відомо, що число *D* – дискримінант деякого квадратного рівняння. Визначити кількість коренів цього рівняння:

- два корені;
- один корінь;
- немає коренів.

Варіант 3.

1. Обчислити значення функції $f(x,y)=\begin{cases} x^2+y, якщо x-парне, \\ x-y, якщо x-непарне. \end{cases}$
2. Відомі ширина основи *a* та висота *h* деякого листа паперу. Визначити орієнтацію цього листа відносно його основи:

- книжкова;
- альбомна;
- лист квадратний.

Варіант 4.

1. Обчислити значення функції $f(x,y)=\begin{cases} x^2-1, якщо y=0, \\ x+2y^2, якщо y\neq 0. \end{cases}$
2. Задано три сторони трикутника $a<b<c$. Визначити, чи є цей трикутник:

- гострокутним;
- прямокутним
- тупокутним.

Варіант 5.

1. Обчислити значення функції $f(x,y)=\begin{cases} y-2x, якщо x<0, \\ \sqrt{x}, якщо x\geq 0. \end{cases}$
2. Тіло, що має щільність ρ_1, помістили в рідину щільності ρ_2. Визначити як буде поводитись тіло:

- плаватиме на поверхні рідини;
- вільно плаватиме всередині рідини;
- опуститься на дно.

Варіант 6.

1. Обчислити значення функції $f(x,y)=\begin{cases} -x+y, якщо y>0, \\ x+\sqrt{y}, якщо y\leq 0. \end{cases}$
2. Відомо, що дата атестації в поточному місяці задається числом D. Визначити положення дати атестації відносно сьогоднішньої дати:

- вже пройшла;
- відбувається сьогодні;
- планується в майбутньому.

Варіант 7.

1. Обчислити значення функції $f(x,y)=\begin{cases} x^2+y, якщо y - непарне, \\ x^2-y, якщо y - парне. \end{cases}$
2. Задано довжини двох відрізків AB та CD. Визначити, як ці відрізки співвідносяться між собою:

- перший відрізок більше другого;
- другий відрізок більше першого;
- відрізки рівні.

Варіант 8.

1. Обчислити значення функції $f(x,y)=\begin{cases} 2x-3y, якщо x=y, \\ 3x+2y, якщо x\neq y. \end{cases}$
2. Деяке тіло рухається прямолінійно з прискоренням a. Визначити, що відбувається з тілом:

- розганяється;
- рухається рівномірно;
- гальмує.

Варіант 9.

1. Обчислити значення функції $f(x,y)=\begin{cases} x^2+y^2, якщо x\cdot y>0, \\ x+2y, якщо x\cdot y\le 0. \end{cases}$
2. Відомо, що найбільший кут трикутника дорівнює α. Визначити, чи є цей трикутник:

- гострокутним;
- прямокутним
- тупокутним.

Варіант 10.

1. Обчислити значення функції $f(x,y)=\begin{cases} x^2+1, якщо x>y, \\ y^2-1, якщо x\le y. \end{cases}$
2. Задані радіуси двох кіл R_1 та R_2. Визначити як співвідносяться ці кола, якщо їхні центри збігаються:

- перше коло міститься всередині другого;
- друге коло міститься всередині першого;
- кола збігаються.

Варіант 11.

1. Обчислити значення функції $f(x,y)=\begin{cases} y+x, якщо x=0, \\ x^2+y, якщо x\ne 0. \end{cases}$
2. За відомим значенням небесної широти деякої зірки визначити в якій півкулі вона знаходиться:

- у північній півкулі;
- на небесному екваторі;
- у південній півкулі.

Варіант 12.

1. Обчислити значення функції $f(x,y)=\begin{cases} \sin(x), якщо x<y, \\ x+y-1, якщо x\ge y. \end{cases}$
2. Парабола задається рівнянням $ax^2+bx+c=0$. Визначити напрямок віток параболи в залежності від значення коефіцієнту a:

- направлені вгору;
- парабола вироджена;
- направлені вниз.

Лабораторна робота №4
JavaScript. Структури даних. Організація взаємодії з користувачем

Завдання базового рівня.

1. Написати код на JavaScript, за яким запитується ім'я, прізвище та вік. В результаті має виводитися вікно з повідомленням «Вітаю, [Ім'я] [Прізвище]! та «Ви повнолітній (неповнолітній)» в залежності від введеного віку.

2. Написати код на JavaScript, за яким обчислюється та виводиться сума чисел від 1 до N (N – задається користувачем).

3. Написати функцію на JavaScript, за якою для двох заданих користувачем чисел виводяться парні числа, що потрапляють до проміжку між заданими числами.

4. Написати код на JavaScript, за яким формується масив з 6 випадкових цілих чисел від 0 до 20. Після цього в консоль браузера спочатку виводяться всі елементи масиву, а потім ті елементи масиву, які більші за 3, але менші за 10.

5. Написати код на JavaScript, за яким формується масив з 20 випадкових цілих чисел від 0 до 10. Після цього перевіряється наявність у масиві числа 4. Якщо це число є в масиві, виводиться вікно з повідомленням «Ура! Воно знайшлося!». Якщо ні – нічого не виводиться. Сам сформований масив має бути виведений в вікно браузера (document.write()).

6. Написати код на JavaScript, за яким формується випадковий пароль. Довжину паролю має задавати користувач. Передбачити різні набори символів, з яких буде формуватися пароль: маленькі літери, маленькі та великі літери, літери та цифри.

Завдання підвищеного рівня.

Гра на вгадування слів «Поле чудес».
Гра має бути реалізована з використанням діалогових вікон (alert, prompt, confirm).
Гра полягає в наступному. Гравець-комп'ютер обирає слово з наперед заданого переліку, а гравець-людина відгадує це слово по літерах.
Наприклад, якщо гравець-комп'ютер загадав слово ІНФОРМАТИКА, то за програмою виводиться відображення шаблону заданого слова з 11 «порожніми місцями», по одному на кожну літеру слова «_ _ _ _ _ _ _ _ _ _ _» та пропонується гравцю-людині ввести літеру.

Кожного разу, коли літера вгадана (літера, вказана гравцем-людиною, є в загаданому слові), виводиться шаблон слова з розміщеними на своїх позиціях вгаданими літерами.
Наприклад, якщо гравець-людина вказав літеру «А», то виводиться «_ _ _ _ _ _ А _ _ _ А». Якщо вказаної літери немає, нараховується штрафний бал. По завершенню (відгадуванню всього слова) виводиться привітання та вказується кількість штрафних балів.

Додатково 1. При роботі програма має враховувати введення як великих, так і маленьких літер.

Додатково 2. Обмежити кількість літер, що можуть бути названі.

Додатково 3. Передбачити виведення попередження про повторне вказування літер.

Лабораторна робота №5
JavaScript. Об'єктна модель документа

Завдання базового рівня.
1. Створити довільну HTML-сторінку, яка буде містити:
Два абзаци тексту, записані в контейнері з визначеними класами, для яких задане довільне форматування.

Маркований список з визначеним ідентифікатором, для якого задане довільне форматування.
7 записів маркованого списку з визначеним однаковим для всіх класом, для якого задане довільне форматування.

Три блоки , що будуть містити текстову інформацію, з власними ідентифікаторами, для яких налагоджене форматування кольору тексту та фону.

2. Написати код на JavaScript, за яким відбувається зміна форматування абзацу ( ) при натисненні лівої кнопки миші на ньому.

3. Написати код на JavaScript, за яким відбувається зміна форматування елементів маркованого списку, таким чином:

- якщо натиснення відбувається на першому елементі списку, то має змінюватися форматування всіх елементів списку;
- якщо натиснення відбувається на другому елементі списку, то має змінюватися форматування лише парних елементів списку;
- якщо натиснення відбувається на третьому елементі списку, то має змінюватися форматування лише першого елементу списку;
- якщо відбувається наведення курсора миші на останній елемент списку, то має виводитися вікно повідомлень з текстом, що відповідає вмісту цього елемента списку;

4. Написати код на JavaScript, за яким відбувається зміна форматування блоків при натисненні лівої кнопки миші на першому з них, таким чином: через певний проміжок часу форматування блоків має циклічно змінюватися (другий блок отримує форматування першого, третій блок отримує форматування другого, перший блок отримує форматування третього) зміна має відбуватися одночасно.

Завдання підвищеного рівня.
1. Створення анімації.
Написати код на JavaScript за яким буде реалізовано рух заголовку h1 по квадрату (спочатку заголовок переміщується на 200 пікселів вправо, потім на 200 пікселів вниз, потім на 200 пікселів вліво, потім на 200 пікселів вгору).

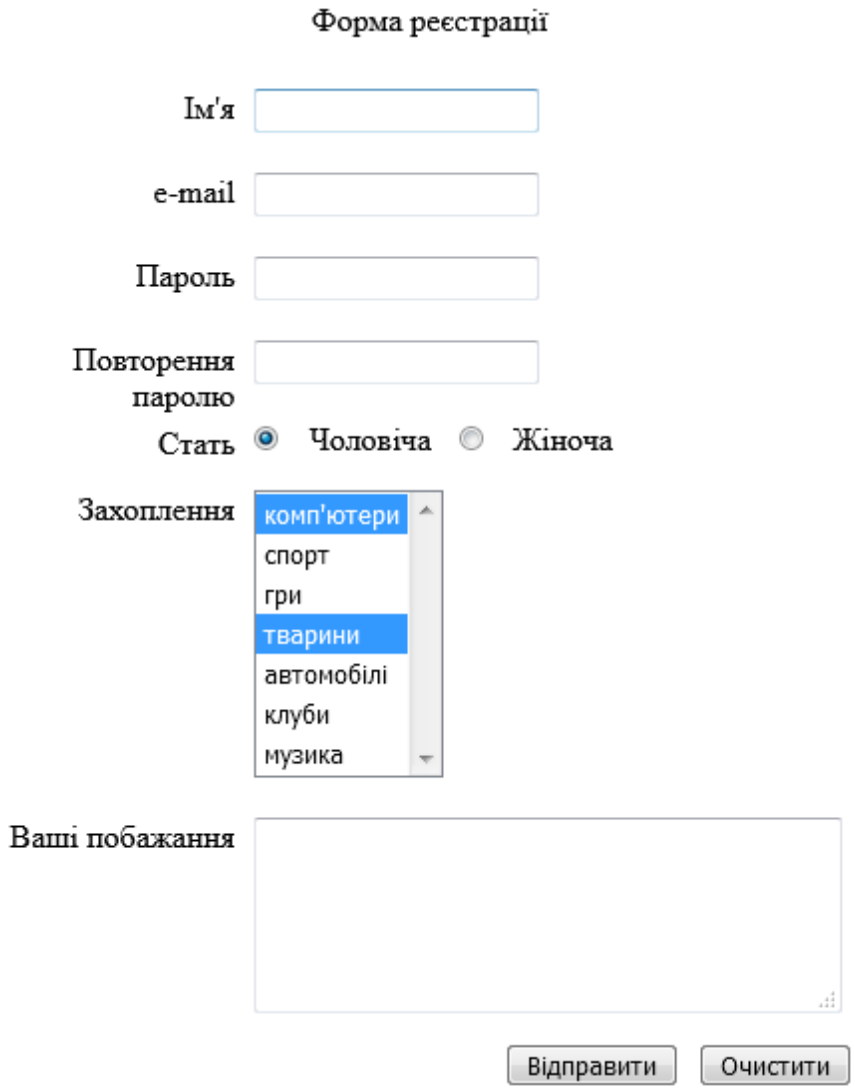
Додатково 1. Передбачити можливість зупинки руху заголовку при натисненні лівої кнопки миші на ньому.

2. Гра на вгадування слів «Поле чудес».
Змінити реалізацію гри «Поле чудес», розробленої при виконанні попередньої лабораторної роботи. Гра має бути реалізована без використання діалогових вікон (alert, prompt, confirm) та елементів управління (input, button, select, textarea, …)

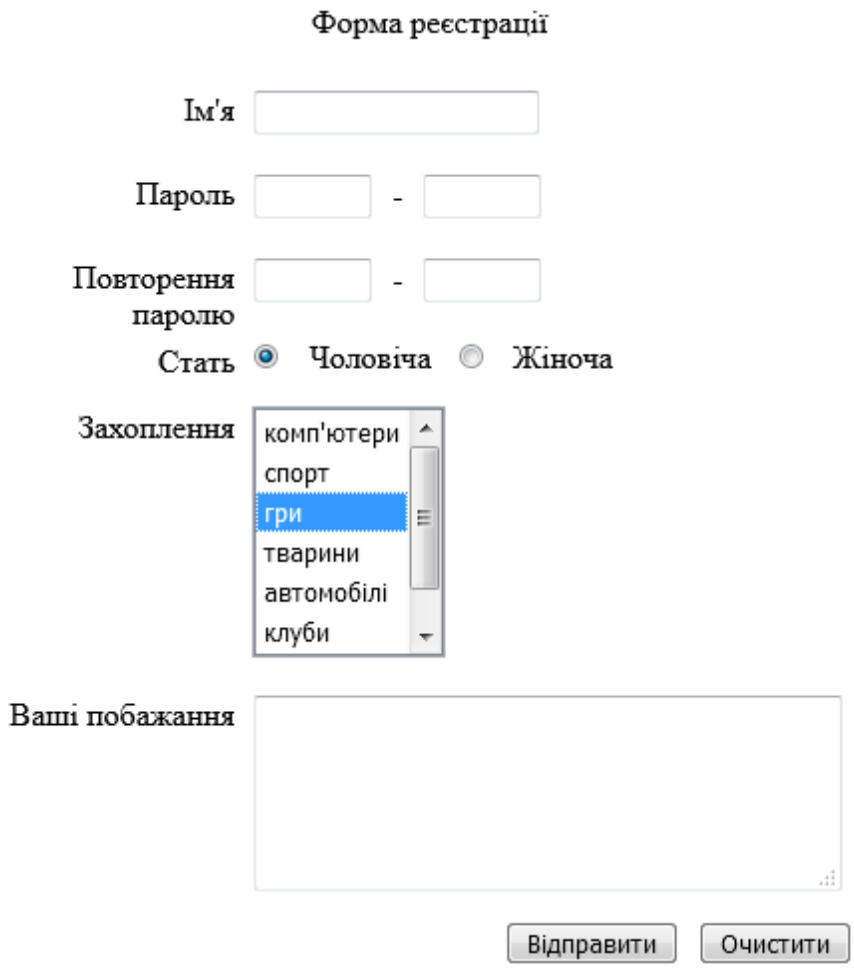
Лабораторна робота №6
JavaScript. Створення форм в HTML-документі

Завдання. Створити форму відповідно до зразка.
Варіанти завдань для самостійного виконання

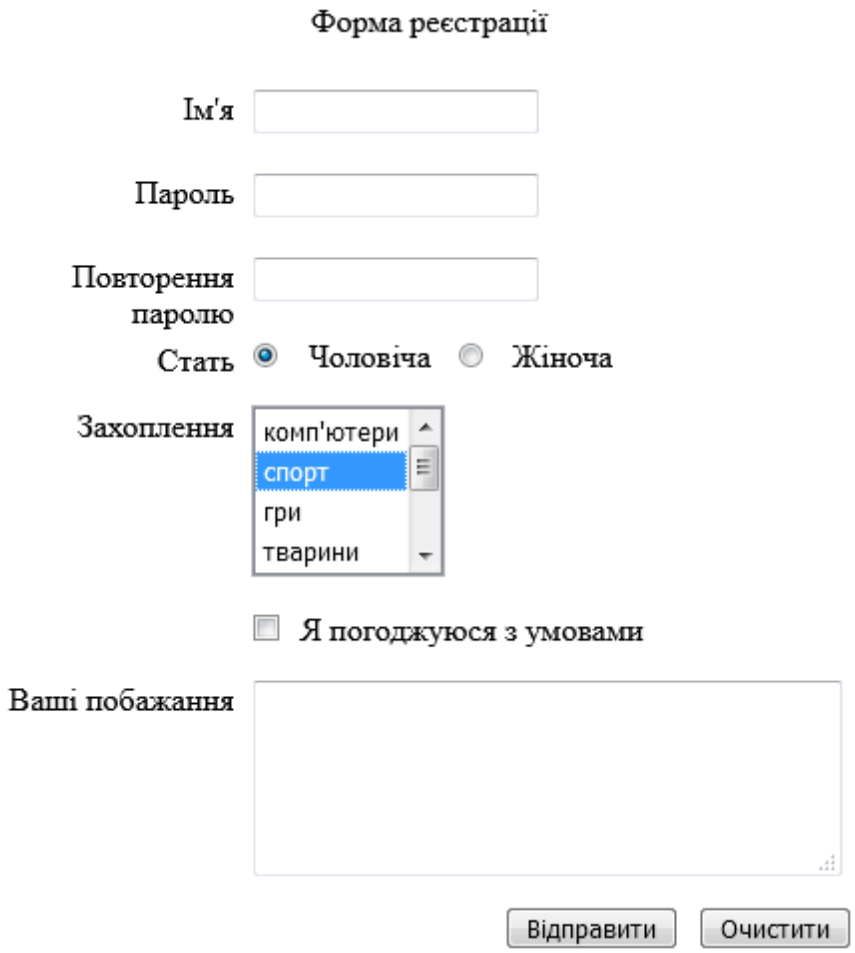
Варіант 1.



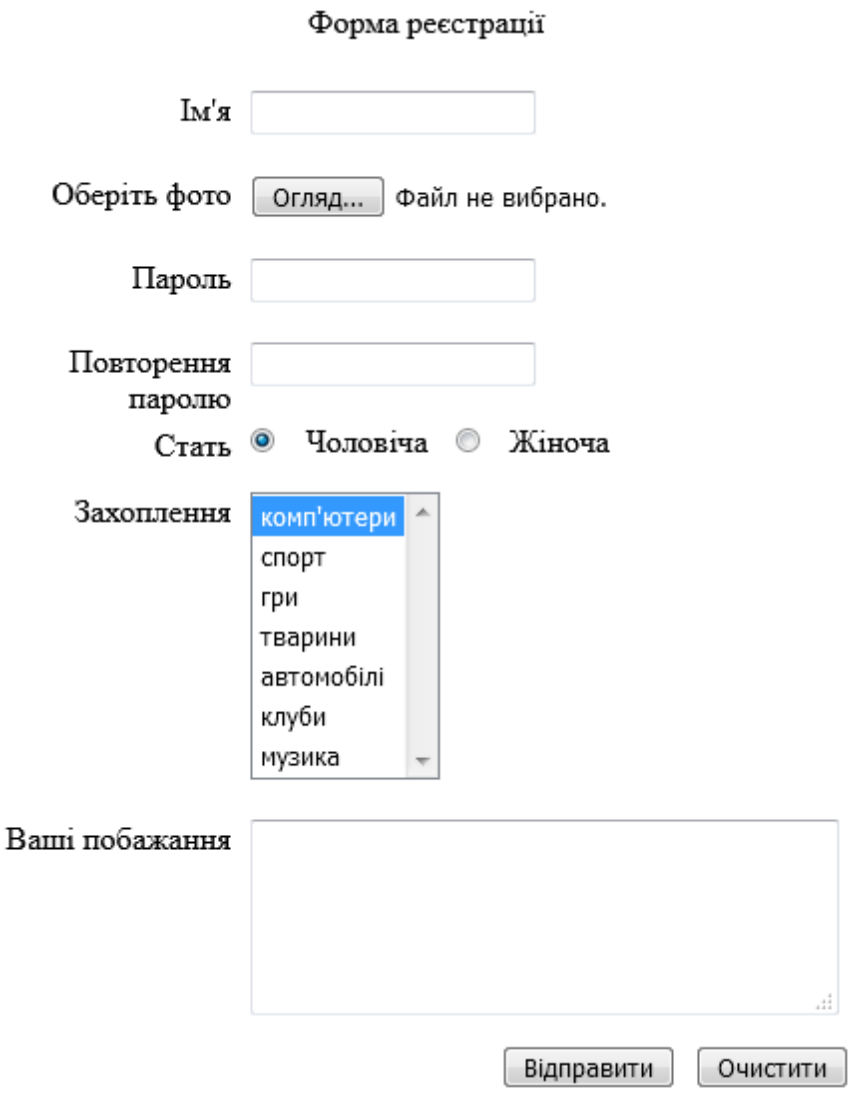
Варіант 2.


Форма реєстрації
Ім'я
Пароль
-
Повторення паролю
-
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
ігри
тварини
автомобілі
клуби
Ваші побажання
Відправити
Очистити

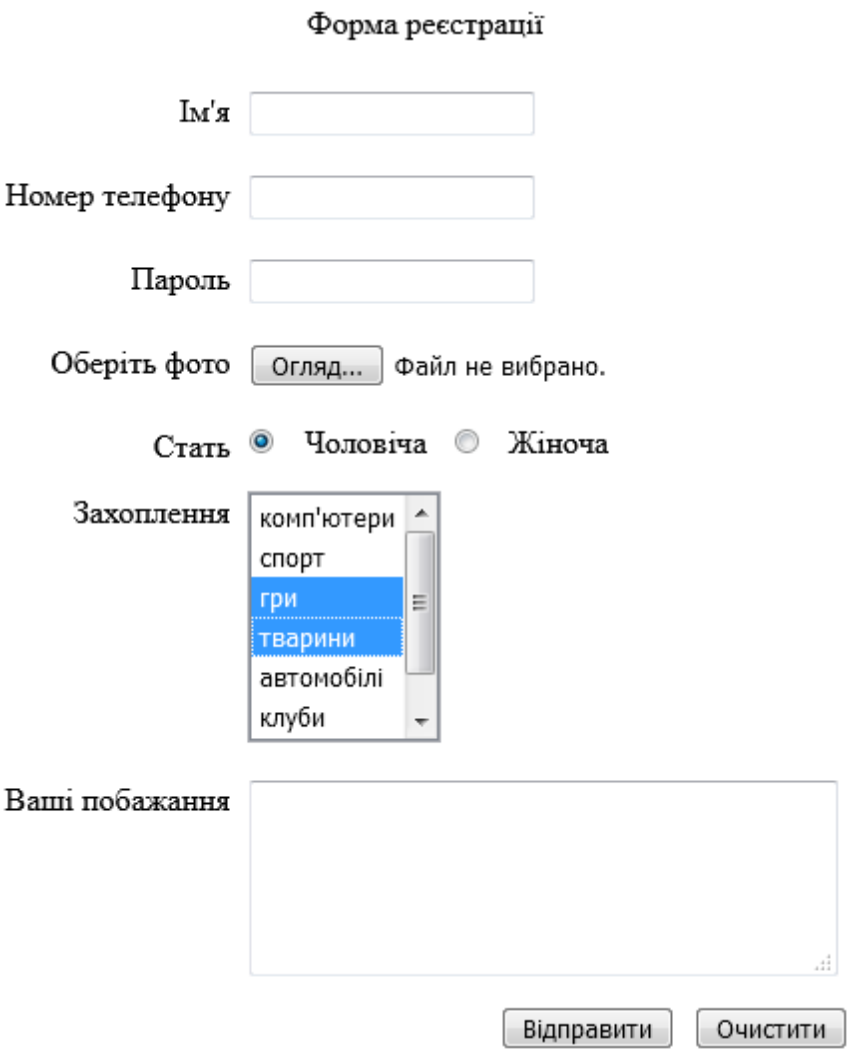
Варіант 3.


Форма реєстрації
Ім'я
Пароль
Повторення паролю
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
ігри
тварини
Я погоджуюся з умовами
Ваші побажання
Відправити
Очистити

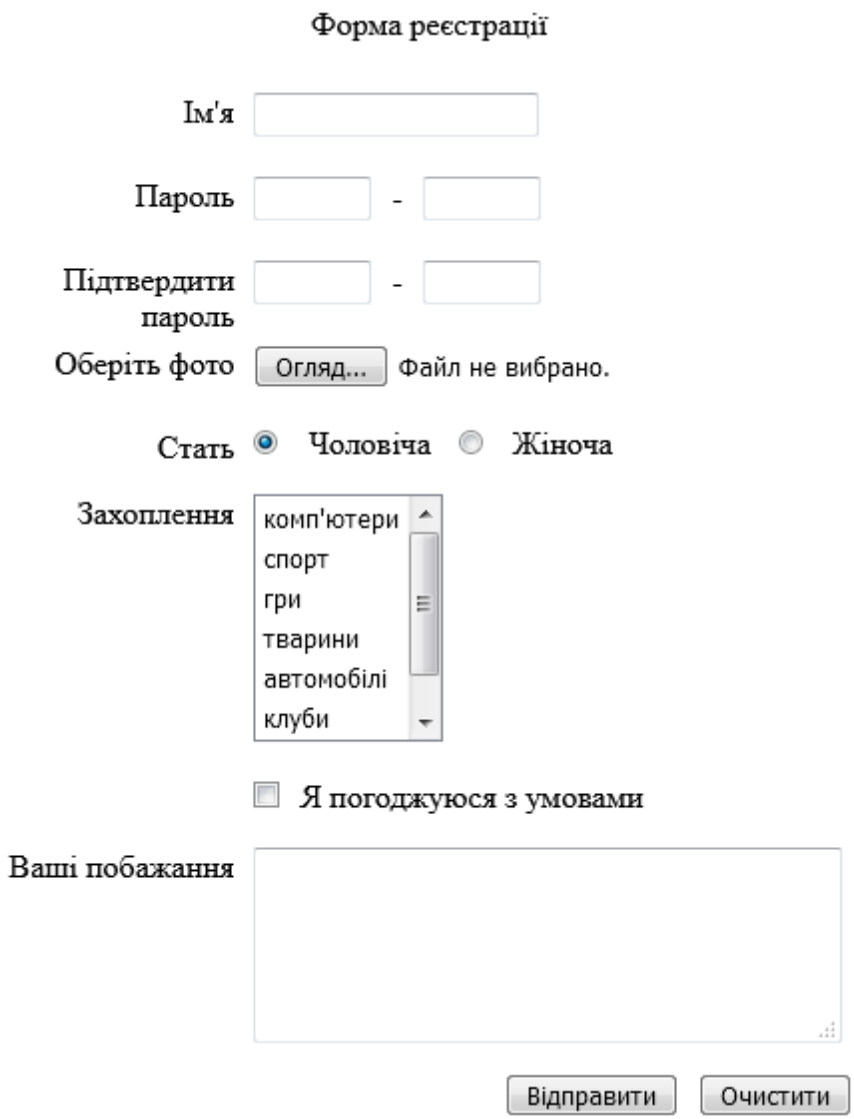
Варіант 4.


Форма реєстрації
Ім'я
Оберіть фото
Огляд...
Файл не вибрано.
Пароль
Повторення паролю
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
гри
тварини
автомобілі
клуби
музика
Ваші побажання
Відправити
Очистити

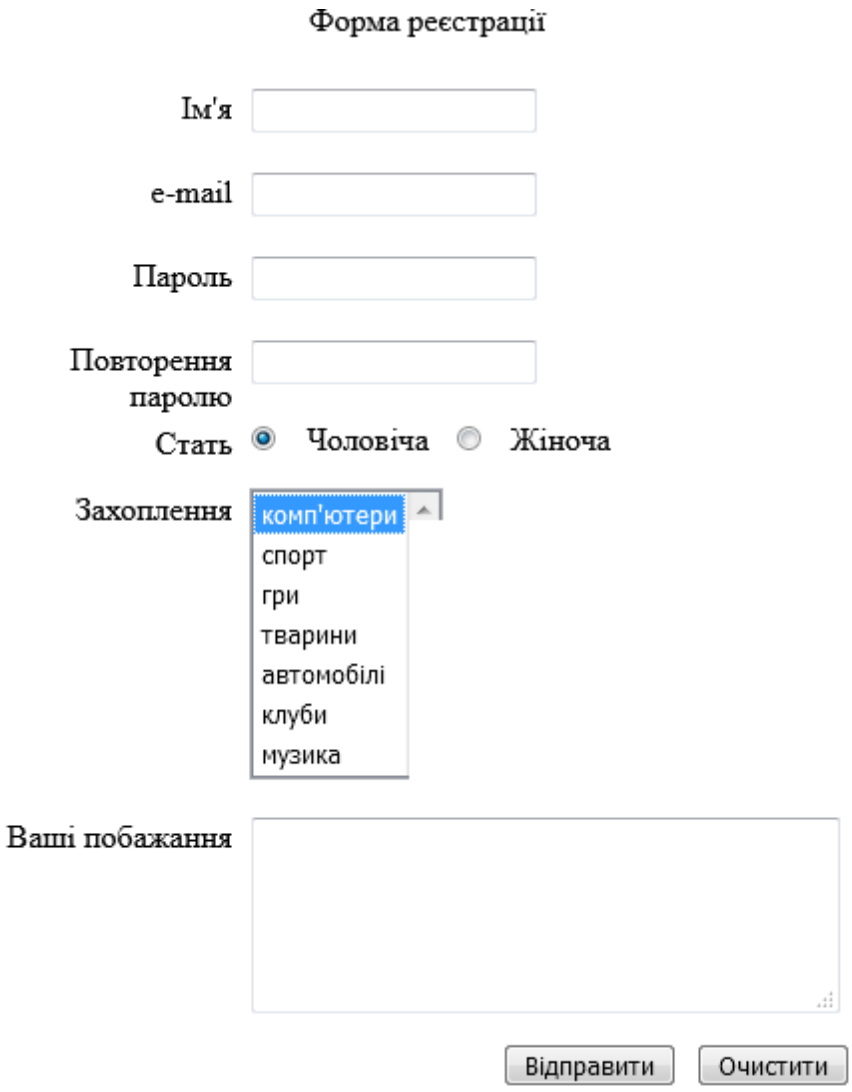
Варіант 5.


Форма реєстрації
Ім'я
Номер телефону
Пароль
Оберіть фото
Огляд...
Файл не вибрано.
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
гри
тварини
автомобілі
клуби
Ваші побажання
Відправити
Очистити

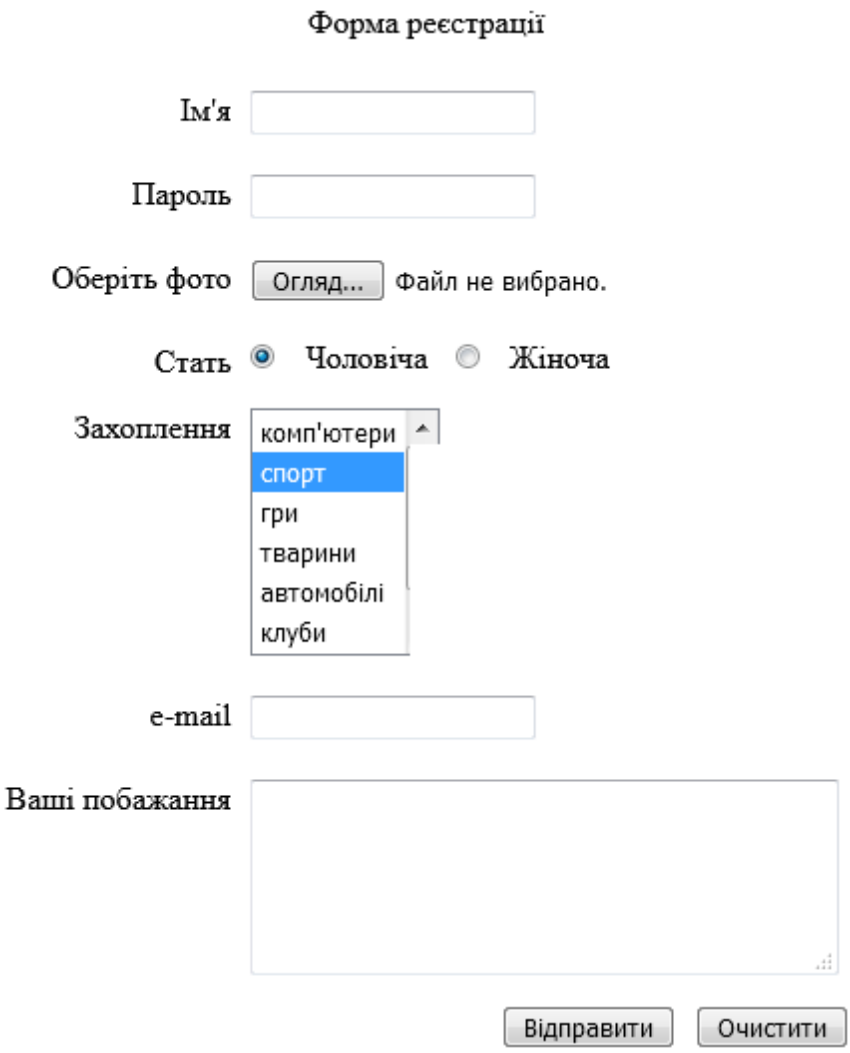
Варіант 6.


Форма реєстрації
Ім'я
Пароль
-
Підтвердити пароль
-
Оберіть фото
Огляд...
Файл не вибрано.
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
гри
тварини
автомобілі
клуби
Я погоджуюся з умовами
Ваші побажання
Відправити
Очистити

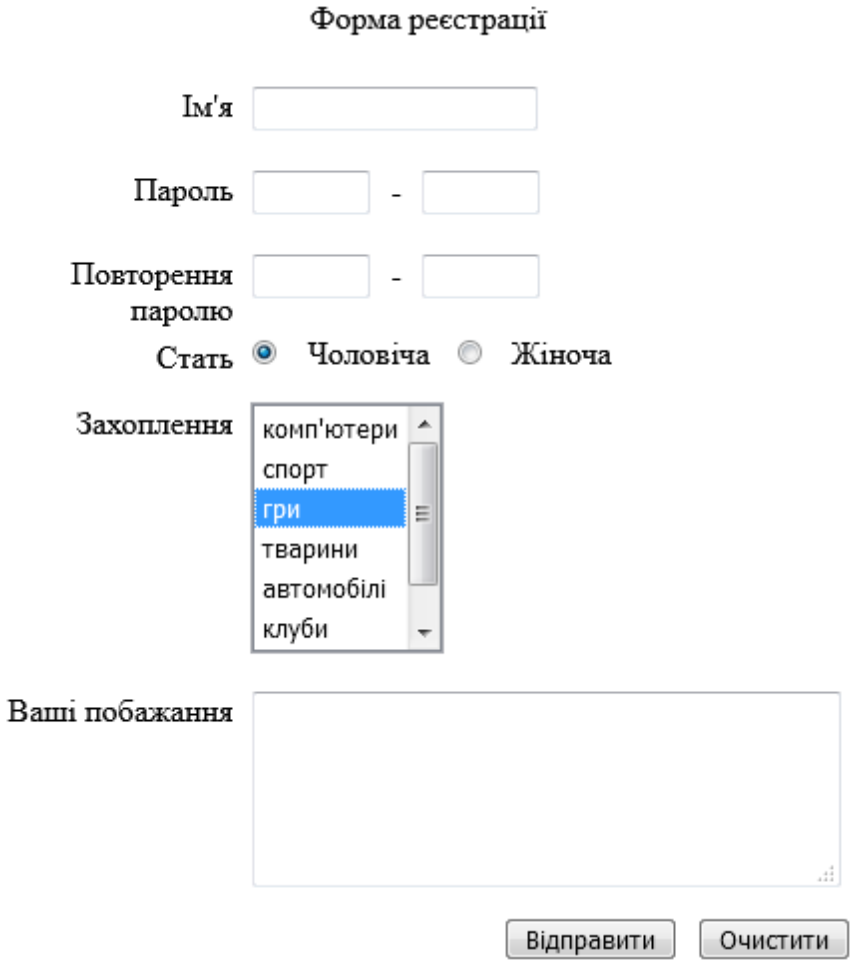
Варіант 7.


Форма реєстрації
Ім'я
e-mail
Пароль
Повторення паролю
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
гри
тварини
автомобілі
клуби
музика
Ваші побажання
Відправити
Очистити

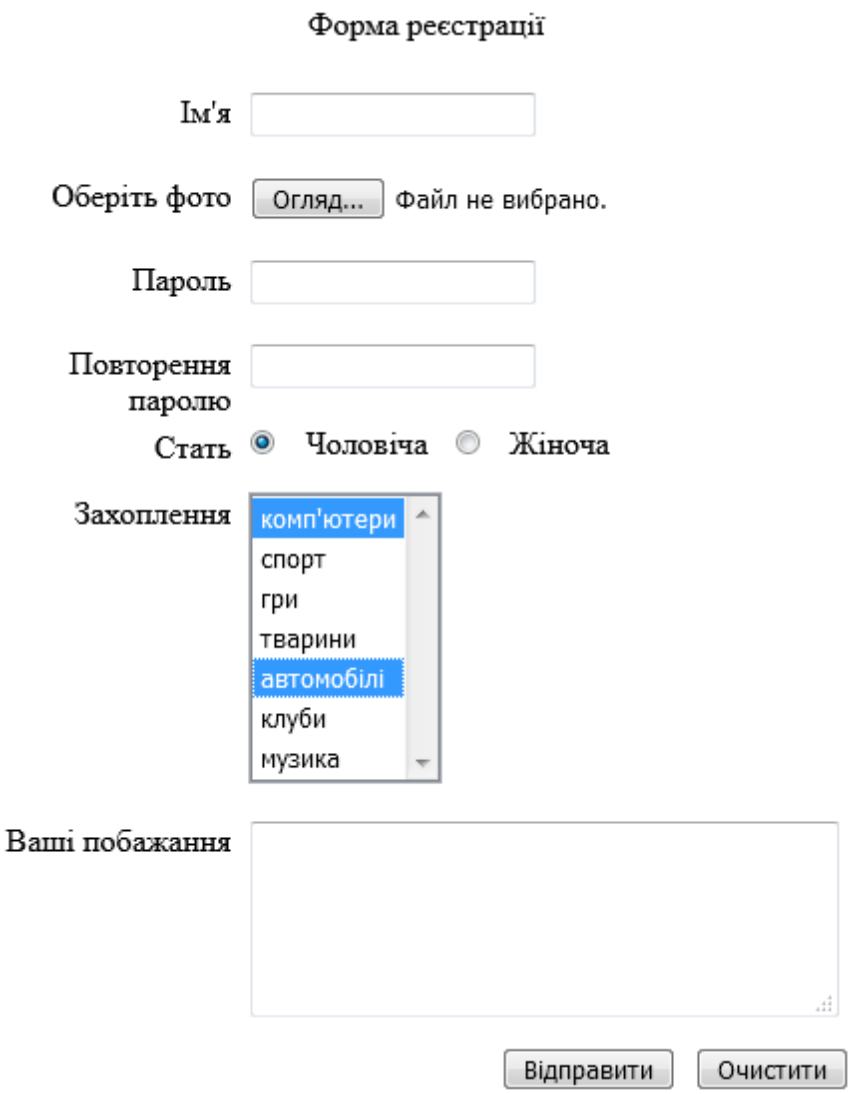
Варіант 8.


Форма реєстрації
Ім'я
Пароль
Оберіть фото
Огляд...
Файл не вибрано.
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
ігри
тварини
автомобілі
клуби
e-mail
Ваші побажання
Відправити
Очистити

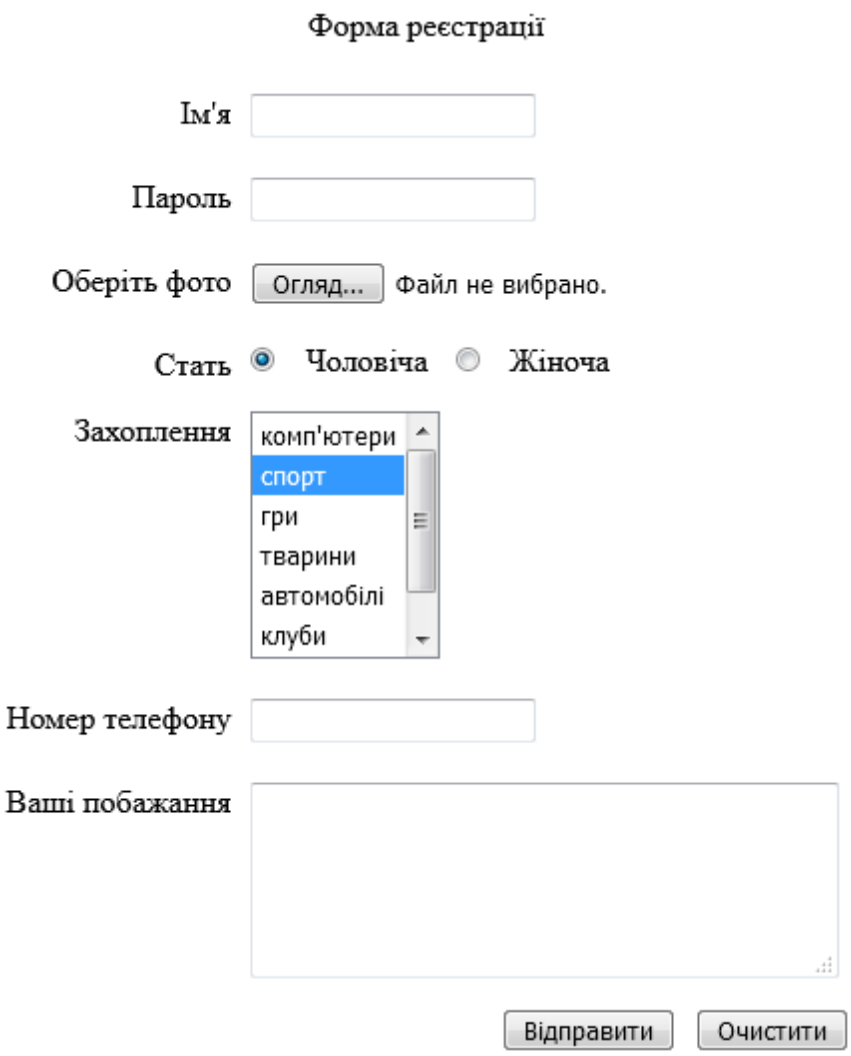
Варіант 9.


Форма реєстрації
Ім'я
Пароль
-
Повторення паролю
-
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
гри
тварини
автомобілі
клуби
Ваші побажання
Відправити
Очистити

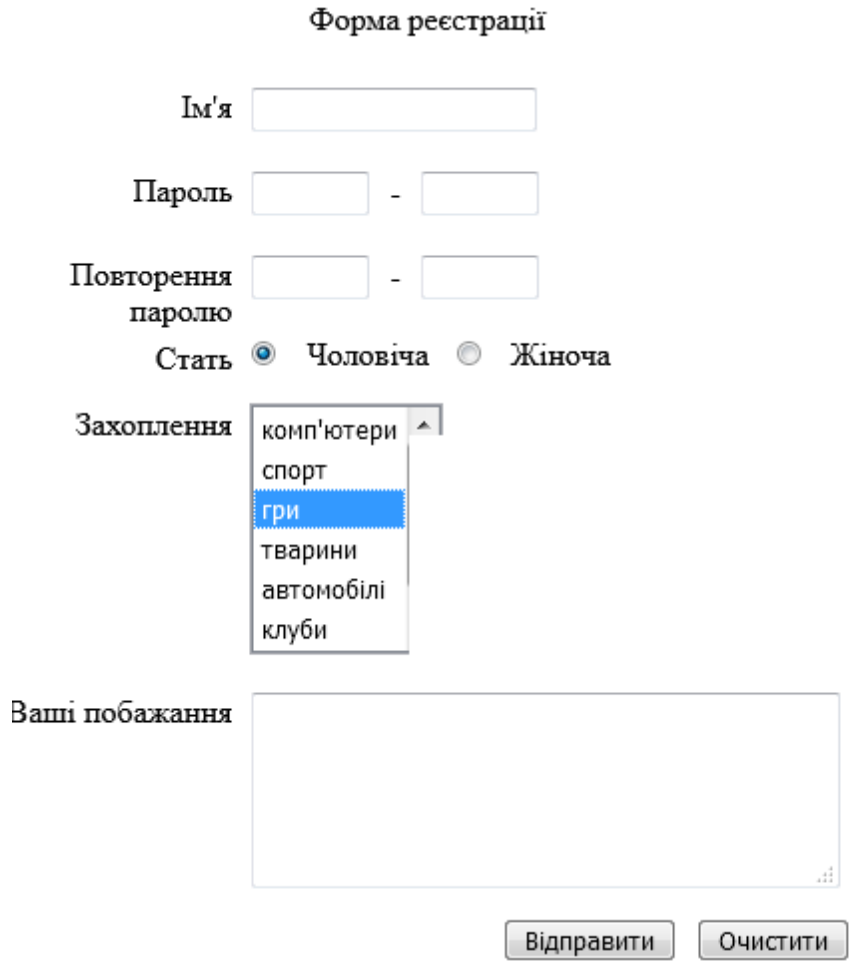
Варіант 10.


Форма реєстрації
Ім'я
Оберіть фото
Огляд...
Файл не вибрано.
Пароль
Повторення паролю
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
гри
тварини
автомобілі
клуби
музика
Ваші побажання
Відправити
Очистити

Варіант 11.


Форма реєстрації
Ім'я
Пароль
Оберіть фото
Огляд...
Файл не вибрано.
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
ігри
тварини
автомобілі
клуби
Номер телефону
Ваші побажання
Відправити
Очистити

Варіант 12.


Форма реєстрації
Ім'я
Пароль
-
Повторення паролю
-
Стать
Чоловіча
Жіноча
Захоплення
комп'ютери
спорт
гри
тварини
автомобілі
клуби
Ваші побажання
Відправити
Очистити

Лабораторна робота №7
JavaScript. Робота з формою в HTML-документі

Завдання. Доповнити форму, розроблену в лабораторній роботі 6, такими можливостями.
1. Перевірка введеного імені має відбуватися після виходу з поля для введення імені. Якщо це ім'я вже раніше було використане (перевіряється за вмістом наперед заданого масиву імен), необхідно повідомити про це користувача:

для варіантів: 1, 4, 7, 10 - вивівши поряд з полем введення імені текст “Допустиме” або “Недопустиме”;
для варіантів: 1, 5, 8, 11 - вивівши поряд з полем введення імені картинку з червоним хрестиком або зеленою галочкою;
для варіантів: 3, 6, 9, 12 - підсвітивши поле введення імені зеленою або червоною рамкою.

2. При необхідності вибору статі має змінюватись фон сторінки (для чоловічої - синій, для жіночої - рожевий). Фон за замовчуванням має відповідати статі за замовчуванням.

3. При необхідності погодитися з умовами, колір тексту визначеного елементу має змінюватися на зелений при встановленні галочки і знову ставати чорним при її прибиранні.

4. При необхідності введення e-mail, перевіряти її на правильність формату після виходу з поля для введення пошти. У разі помилкового введення змінювати колір введеної пошти на червоний.

5. При необхідності введення номеру телефону, перевіряти його на правильність формату (---)------- після виходу з поля для введення номеру телефону. У разі помилкового введення змінювати колір введеного номеру телефону на червоний.

6. Після натиснення на кнопку "Відправити" має відбуватися перевірка введених даних, та в результаті вдалої перевірки виводитися повідомлення з привітанням (наприклад: "Шановний користувач ..."), вмістом всіх полів та зроблених виборів. При виявленні даних, що не пройшли перевірку, вивести відповідне повідомлення.
При перевірці:
передбачити перевірку заповнення всіх полів, дозволу використання введеного імені та виконаного вибору;
при необхідності введення паролю та його підтвердження передбачити збіг відповідних полів;

при наявності вибору статі сформувати текст привітання відповідно до вказаної статі.

7. Після натиснення кнопки "Очистити" необхідно всі поля привести до початкового стану.

Лабораторна робота №8
JavaScript. Внесення змін в HTML-документ

Завдання базового рівня.

1. Створити документ, що містить кнопку «Додати блоки». Написати код на JavaScript, за яким при натисненні на кнопку «Додати блоки» в документ додаються 20 елементів div. Парні блоки повинні мати рамку червоного кольору, непарні – блакитного кольору.

2. Створити документ, що містить блок та кнопку «Додати зображення». Написати код на JavaScript, за яким при натисненні на кнопку «Додати зображення» в документ додаються всі зображення з деякого масиву з посиланнями на зображення.

3. Доповнити лабораторну роботу HTML №4 наступними можливостями. Після вдалої перевірки за натисненням на кнопку "Відправити" введене ім’я має додаватися в кінці документу:
для парних варіантів – до нумерованого списку,
для непарних варіантів – до елементу вибору Select.

4. Створити документ з двома кнопками на сторінці: «Додати блоки», «Вилучити блок». За натисненням на кнопку «Додати блоки» запитується кількість блоків для додавання (від 1 до 6) і додається вказана кількість блоків. За натисненням на кнопку «Вилучити блок» запитується порядковий номер блоку, що необхідно вилучити, і при наявності такого блоку він вилучається. Інформація про 6 блоків має зберігатися в масиві об’єктів, окремий об’єкт має структуру:

```
{ title: "текст",
width: "50px",
height: "50px",
imgURL: "pic.jpg",
};
```

Додатково. Додати на сторінку кнопку «Пошук». За натисненням на цю кнопку запитується слово для пошуку, і якщо на сторінці є блок, у якого title відповідає шуканому слову, то для цього блоку встановлюється рамка (товщину, колір та фон задати самостійно).

Завдання підвищеного рівня.
Поле для гри в шашки.
Написати скрипти, за якими відбувається відображення шахової дошки, та скрипт розставлення шашок на цій дошці. Приклад вікна браузера після виконання скриптів подано на рисунку 10.8.1

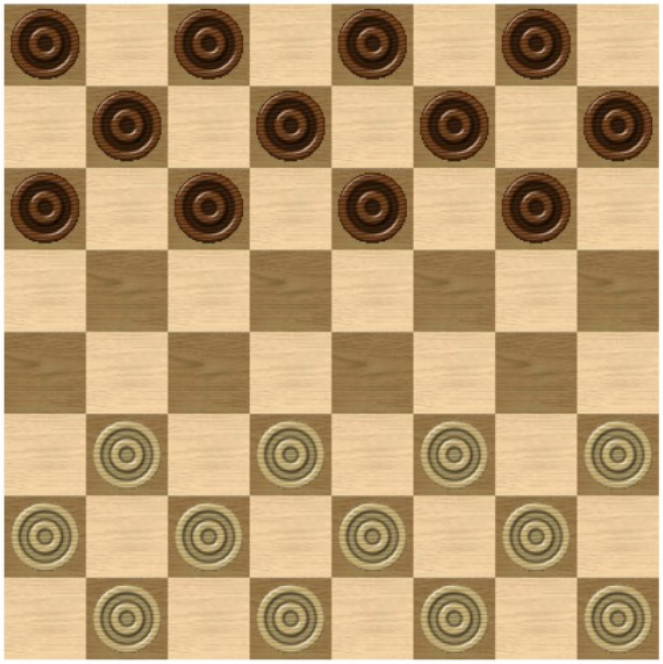


Рис. 10.8.1. Приклад виконання завдання

Структура HTML-документу (змінювати структуру файлу шаблону заборонено):

```
<html>
 <head>
   <style>#board{width:400px;}</style>
</head>
<body>
  <input type="button" value="Намалювати шахову дошку" onclick="">
  <input type="button" value="Розставити шашки"
onclick=""><br><br>
  <div id="board"></div>
</body>
</html>
```

Для побудови шахової дошки та розстановки шашок необхідно використовувати графічні зображення (зазначені зображення можуть бути знайдені в мережі Інтернет або надані викладачем), а саме: окреме зображення **світлої клітинки**; окреме зображення **темної клітинки**; зображення **світлої шашки**; зображення **темної шашки**.

Додатково 1. Написати скрипти, за якими буде відбуватися іменування стовпців та рядків шахової дошки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вступ до вебтехнологій : навч. посіб. для студ. спец. 122 – «Комп'ютерні науки» / розробники : В. І. Пецко, А. С. Беца, Г. В. Мазютинець, О. В. Міца ; рец. : Ф. Е. Гече , Й. І. Головач. – Ужгород : РІК-У, 2024. – 252 с.

2. Сучасний підручник з JavaScript [Електронний ресурс]. Режим доступу до ресурсу: https://uk.javascript.info/

3. Мельник Р. А. Програмування вебзастосувань (фронт-енд та бек-енд). Навчальний посібник. Львів : Видавництво Львівської політехніки, 2018. 248 с.

4. JavaScript Підручник. Основи вебпрограмування [Електронний ресурс]. Режим доступу до ресурсу: https://w3schoolsua.github.io/js/

5. JavaScript: матеріали для самопідготовки [Електронний ресурс]. Режим доступу до ресурсу: https://campus.epam.ua/ua/blog/255

6. Danny Goodman, Michael Morrison, Paul Novitski, Tia Gustaff Rayl. JavaScript Bible. -John Wiley & Sons, 2010 . - 1224 p.

7. Курінний С. Л. Розробка веб сайтів для початківців (html, css, javascript) [Електронний ресурс]. Режим доступу до ресурсу: https://kurin.home.blog/wp-content/uploads/2022/12/ua-kurinny-mozilla-v016.pdf

8. Understanding the JavaScript runtime environment [Електронний ресурс]. Режим доступу до ресурсу: https://medium.com/@gemma.croad/understanding-the-javascript-runtime-environment-4dd8f52f6fca

9. How the web works [Електронний ресурс]. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_the_web_works

10. Uncover the JavaScript: Source Code to Machine Instruction Generation [Електронний ресурс]. Режим доступу до ресурсу: https://medium.com/@misbahulalam/uncover-the-javascript-workflow-of-javascript-engine-646917112013

11. Introduction to JS Engines and Runtimes [Електронний ресурс]. Режим доступу до ресурсу:https://algodaily.com/lessons/introduction-to-js-engines-and-runtimes