Методичні вказівки
до лабораторних робіт з дисципліни
«Технології програмування»

для студентів спеціальності

125 – «Кібербезпека»

Частина 1

Міністерство освіти і науки України
Вінницький національний технічний університет

Методичні вказівки до лабораторних робіт з дисципліни «Технології програмування»

для студентів спеціальності

125 – «Кібербезпека»

Частина 1

Вінниця
ВНТУ
2022

Затверджено до елекронного видання Методичною радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 6 від 17.02.2022 р.)

Рецензенти:

А. А. Шиян, кандидат фізико-математичих наук, доцент

Д. І. Катєльніков, кандидат технічних наук, доцент

Методичні вказівки до лабораторних робіт з дисципліни «Технології програмування» для студентів спеціальності 125 – «Кібербезпека». Частина 1 [Електронний ресурс] / Уклад.: Ю. Є Яремчук, І. О. Дьогтєва, Д. П. Присяжний – Вінниця: ВНТУ, 2022. – 70 с.

У даних методичних вказівках до лабораторних робіт наводяться основні рекомендації до вивчення, підготовки та проведення лабораторних робіт з дисципліни «Технології програмування» та організації самостійної роботи студентів.

ЗМІСТ

ВСТУП

Методичні вказівки до виконання лабораторних робіт призначені для підготовки та виконання лабораторних робіт з дисципліни «Технології програмування» для студентів спеціальності 125 – «Кібербезпека». Частина 1.

Метою методичних вказівок є надання допомоги студентам в отриманні практичних навичок роботиз текстами програм на мові JavaScript у середовищі Visual Studio Code.

У методичних вказівках наведено: необхідні теоретичні відомості з наведеними прикладами коду, завдання, які має виконати студент.

При підготовці до виконання лабораторної роботи студент має ознайомитись з відповідними теоретичними відомостями, ознайомитись з завданнями до лабораторних робіт.

Під час лабораторного заняття студент демонструє викладачеві результати роботи (згідно зі своїми варіантами), проводить консультації з питань, які виникли, та завершує роботу.

Після закінчення виконання кожної роботи студенти складають індивідуальні звіти, що містять відповіді процедури рішень, результати роботи, висновки.

Захист роботи полягає в виконанні завдання до лабораторної роботи, відповіді на питання по темі лабораторної роботи і внесення деяких змін в тексти програм, які розроблялись, в присутності викладача.

Під час виконання лабораторних робіт студенти зможуть:

- працювати в середовищі розробки, з інструментами розробника, браузерами;
- створювати, налагоджувати, виконувати програми;
- створювати і використовувати змінні;
- працювати з типами даних;
- використовувати базові оператори, властивості і методи об’єкта Math;
- використовувати методи JavaScript, що дозволяють виводити діалогові вікна;
- використовувати умовні конструкції, оператори порівняння, логічні оператори;
- працювати з циклами;
- використовувати оператор break, інструкцію continue, мітки;
- працювати з функціями, функціями-колбеками, анонімними функціями.

ЛАБОРАТОРНА РОБОТА № 1
СЕРЕДОВИЩЕ РОЗРОБКИ. МЕТОДИ, ЩО ДОЗВОЛЯЮТЬ ВИВОДИТИ КОРИСТУВАЧЕВІ ДІАЛОГОВІ ВІКНА.

Мета: ознайомитись з інтегрованим середовищем розробки, інструментами розробника; методами JavaScript, що дозволяють виводити діалогові вікна.

1.1 Теоретичні відомості

1.1.1 Середовище розробки

До основних типів редакторів належать: IDE та «легкі» редактори.

Терміном **IDE** (Integrated Development Environment, «інтегроване середовище розробки») називають потужні редактори з рядом функцій, які працюють в рамках цілого проекту. IDE завантажує проект, дозволяє перемикатися між файлами, пропонує автодоповнення за кодом всього проекту (а не тільки відкритого файлу), інтегроване з системою контролю версій (git), середовищем для тестування та іншими інструментами на рівні всього проекту. Наприклад: Visual Studio Code; WebStorm.

На рис. 1.1 продемонстровано сторінку офіційного ресурсу Visual Studio Code, на рис. 1.2 початкова сторінка середовища.

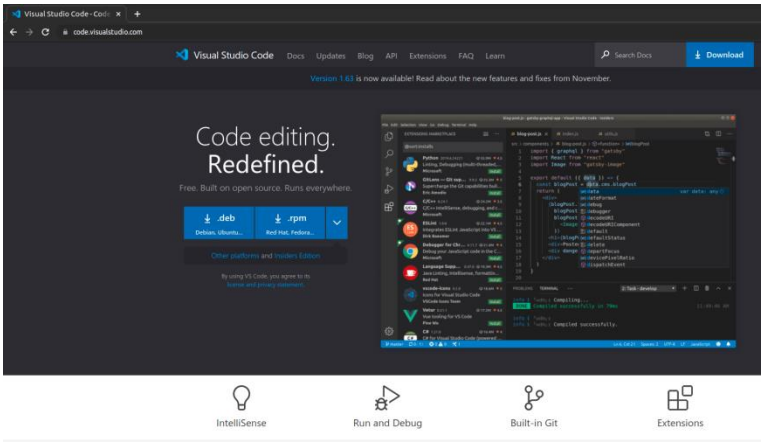

Рисунок 1.1 – Офіційний сайт Visual Studio Code

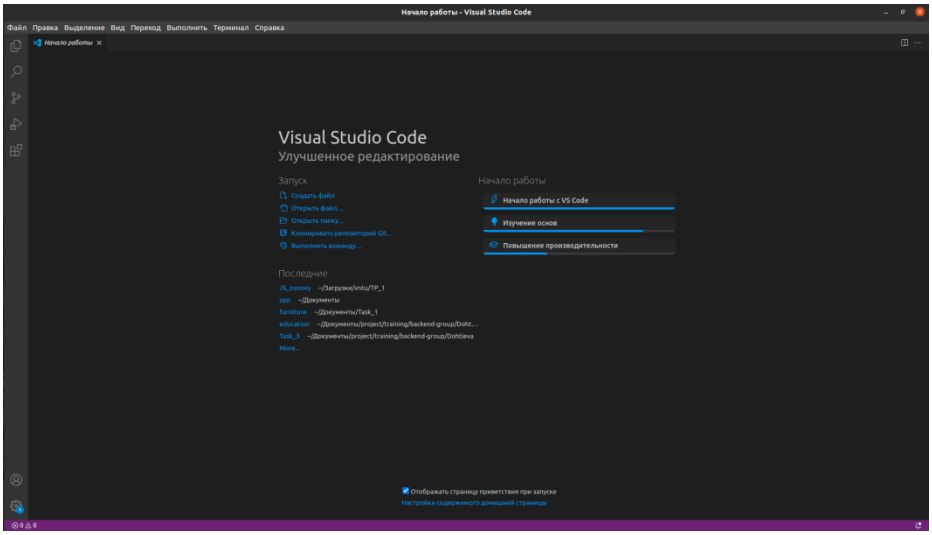

Рисунок 1.2 – Стартова сторінка середовища Visual Studio Code

Для Windows є ще Visual Studio. Visual Studio – це платне потужне середовище розробки, яке працює тільки на Windows. Воно добре підходить для .NET платформи. У даного середовища є безкоштовна версія, яка називається Visual Studio Community.

Головна відмінність між «**легким**» редактором і IDE полягає в тому, що IDE працює на рівні цілого проекту, тому воно завантажує більше даних при запуску, аналізує структуру проекту, якщо це необхідно, тощо. Якщо робота ведеться з невеликою кількістю файлів, зокрема одним, то зручніше відкрити його в «легкому» редакторі.

На практиці «легкі» редактори можуть мати безліч плагінів, включаючи автодоповнення і аналізатори синтаксису на рівні директорії, тому кордони між IDE і «легкими» редакторами досить розмиті.

Серед «легких» редакторів мона виділити:

Atom (багатоплатформовий, безкоштовний);

Sublime Text (багатоплатформовий, умовно-безкоштовний);

Notepad ++ (Windows, безкоштовний), продемонстровано на рис. 1.3;

Vim і Emacs.

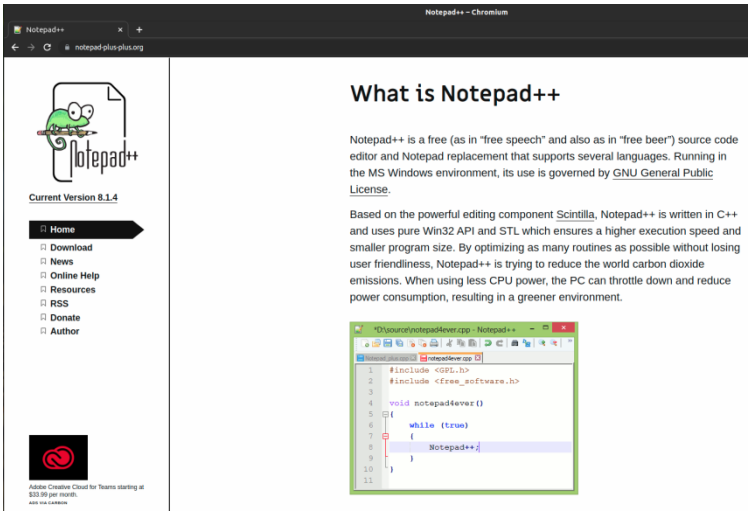

Рисунок 1.3 – Офіційний сайт Notepad ++

Варто також зазначити ряд безкоштовних онлайн редакторів для тестування коду (табл.1.1).

Таблиця 1.1 – Довідкова інформація щодо онлайн редакторів

Назва	Характеристики
Codepen (рис. 1.4)	онлайн-редактор з можливістю спільного редагування; складається з панелі для HTML, CSS, JavaScript, а також вікна для попереднього перегляду в режимі реального часу
JSFiddle (рис. 1.5)	схожий на «пісочницю»; взаємодіє з HTML і CSS, можливість редагувати на сервісі і відразу спостерігати за результатами змін; можна додавати External Requests в бічній панелі, що дозволяє підключати зовнішні JS і CSS файли
JSBin (рис. 1.6)	простіша альтернатива JSFiddle; дозволяє редагувати HTML, CSS і JavaScript, перемикаючись між вкладками на одній сторінці, а також перемикати панелі попереднього перегляду і консолі для максимальної гнучкості; має вбудовані бібліотеки
Liveweave (рис. 1.7)	дозволяє працювати в режимі реального часу і підключатися до певних сторонніх бібліотек, таких як jQuery

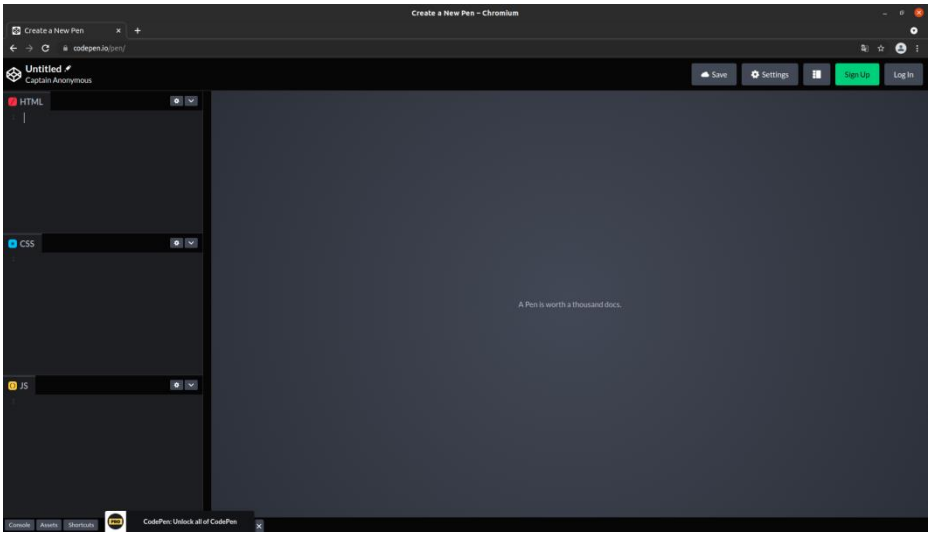

Рисунок 1.4 – Робоча зона Codepen

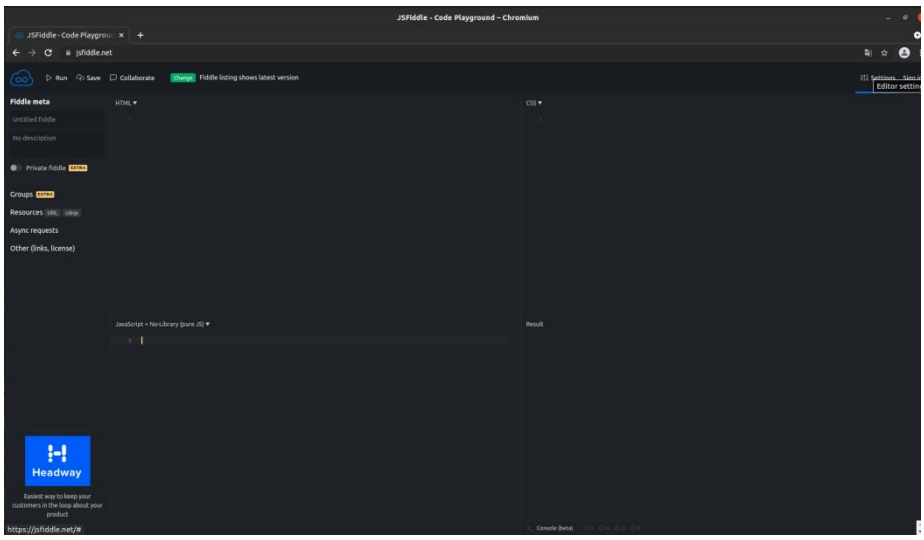

Рисунок 1.5 – Робоча зона JSFiddle

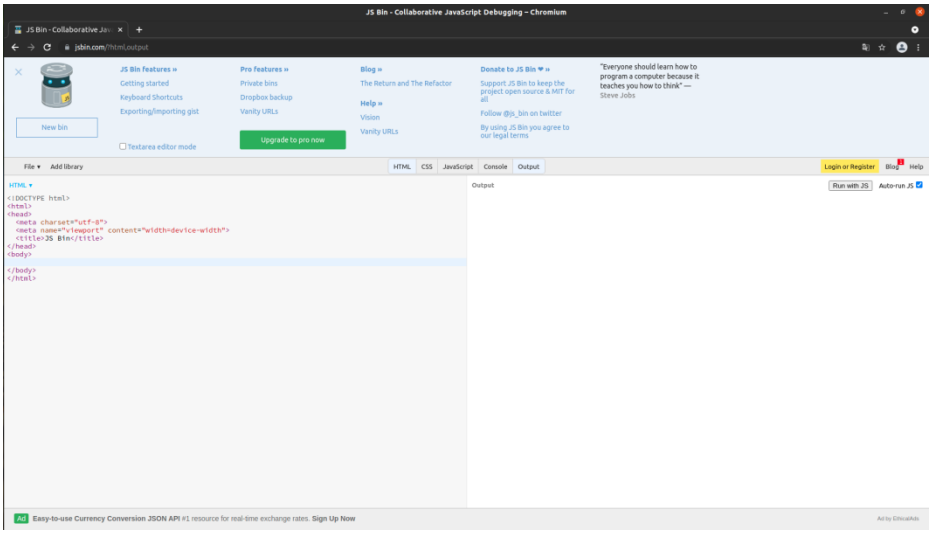

Рисунок 1.6 – Робоча зона JSBin

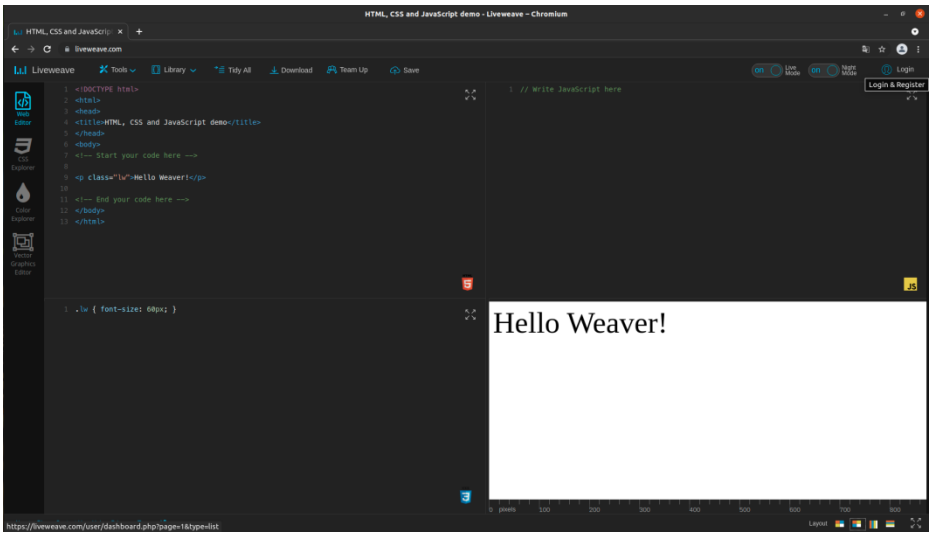

Рисунок 1.7 – Робоча зона Liveweave

Код сторінки з **інструментами розробника** в браузері відкривається: через контекстне меню (рис. 1.8, 1.9);

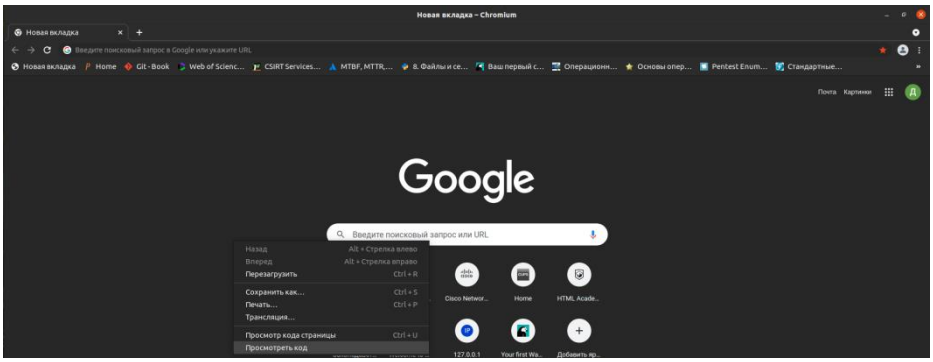

Рисунок 1.8 – Виклик контекстного меню

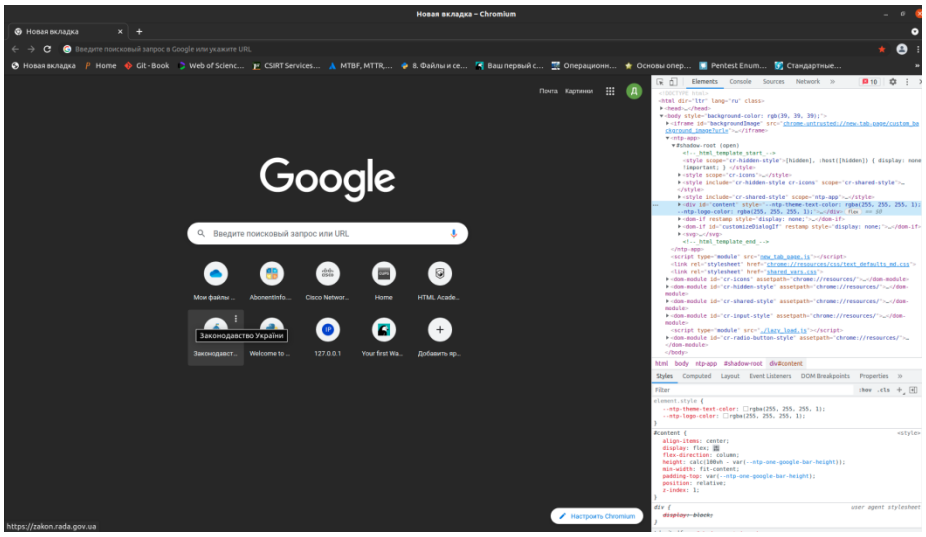

Рисунок 1.9 – Інструмент розробника через виклик контекстного меню

через налаштування (рис. 1.10, 1.11);

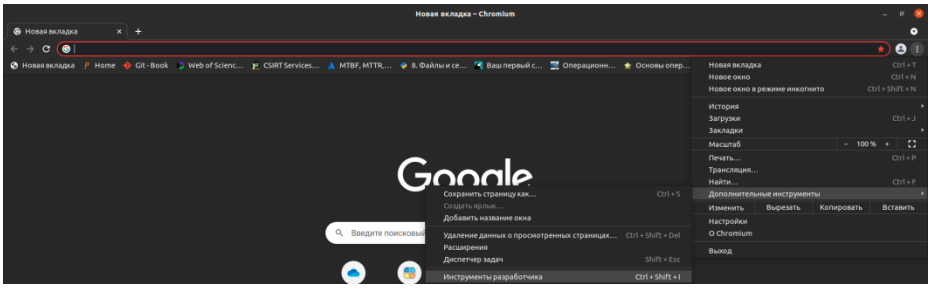

Рисунок 1.10 – Пошук через налаштування

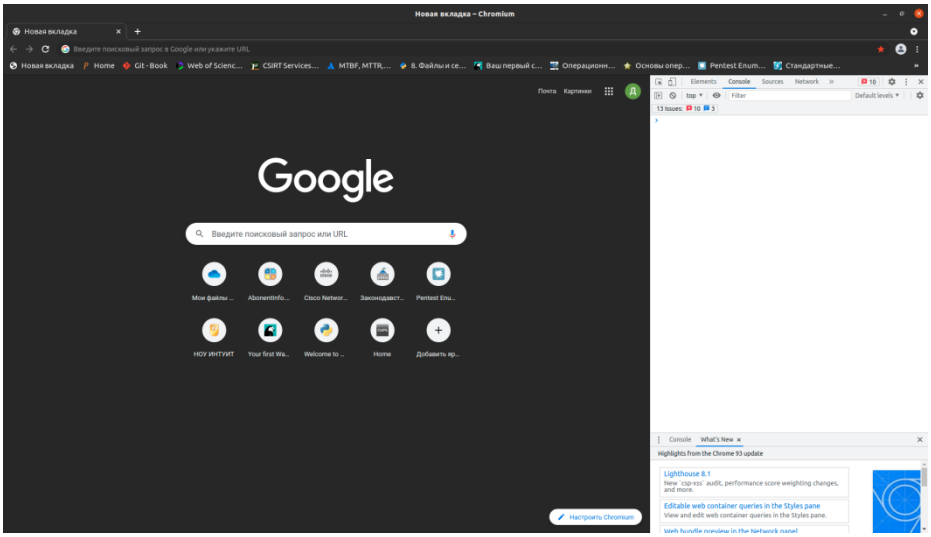

Рисунок 1.11 – Інструмент розробника через налаштування

при натисканні на F12 (в переважній більшості браузерів) або через використання поєднання клавіш, наприклад Ctrl + Shift + I (рис. 1.12).

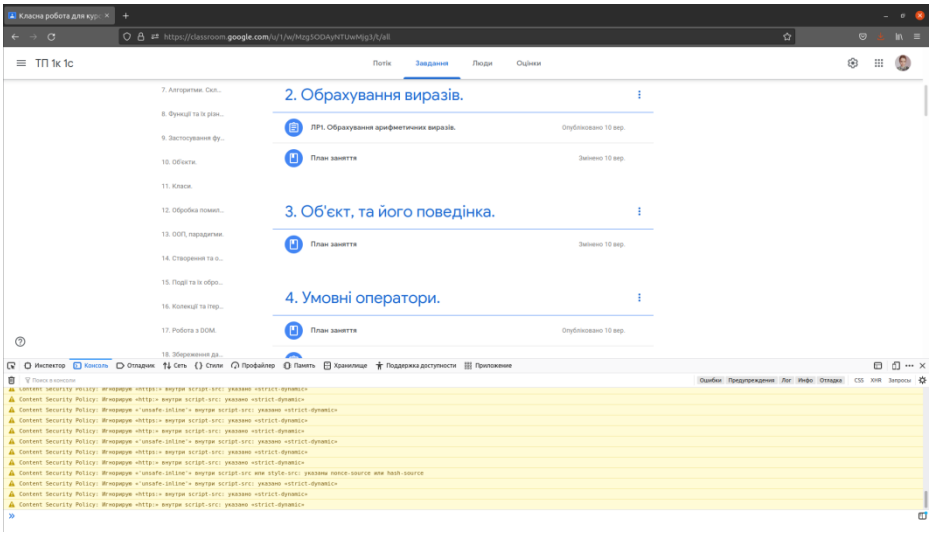

Рисунок 1.12 – Інструмент розробника через натискання відповідних клавіш

1.1.2. Варіанти виконання скрипту

Програмний код розміщується в HTML-сторінках. У загальному випадку можна виділити п'ять способів розміщення коду JavaScript:

в теговому контейнері <BODY> ... </BODY> (рис. 1.13);

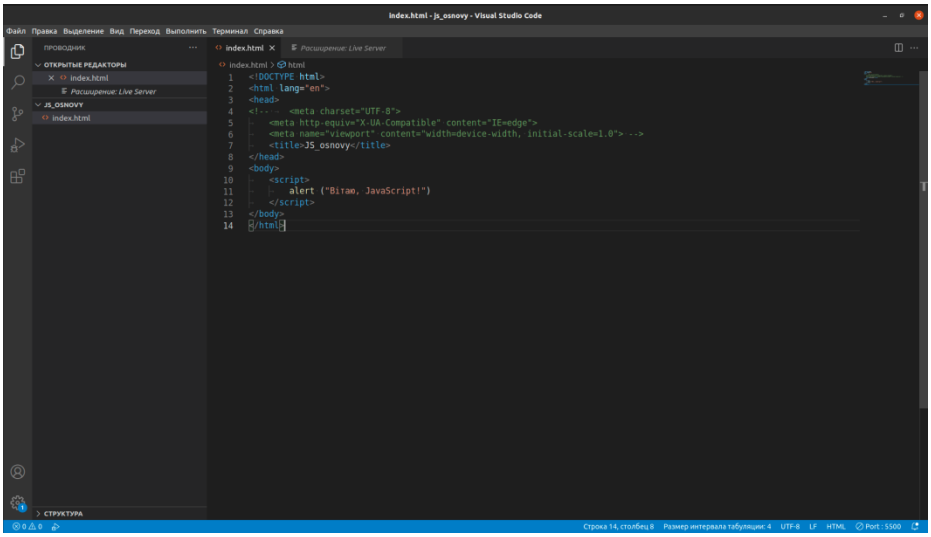

Рисунок 1.13 – Код JavaScript в <BODY> ... </BODY>

в контейнері <HEAD> ... </HEAD> – якщо код скрипта представляє собою функцію, яка відкликається у відповідь на певні події;

у зовнішніх файлах з розширенням .js, звернення до якого здійснюється з використанням тега <SCRIPT> (рис. 1.14), файл скрипта можна підключити до HTML за допомогою атрибута src;

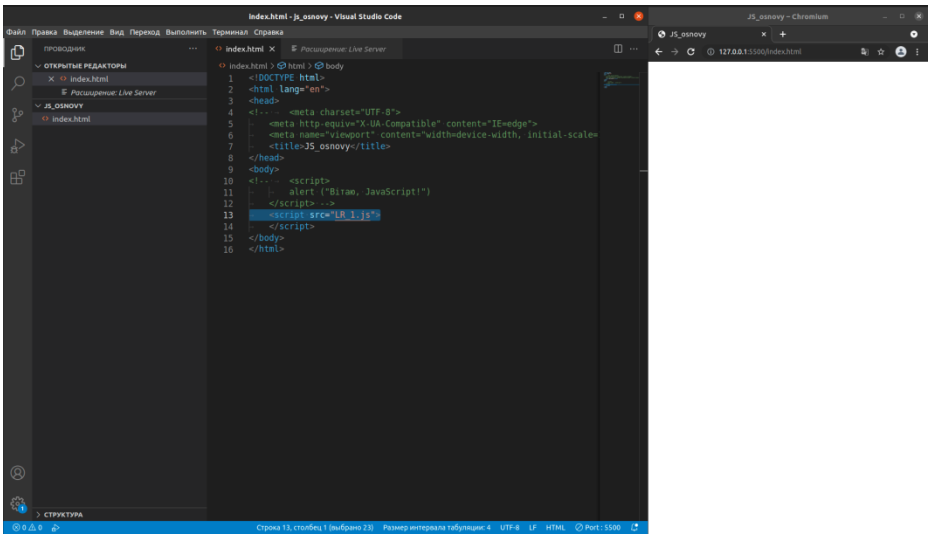

Рисунок 1.14 – Код JavaScript в <SCRIPT> ... </SCRIPT>

гіпертекстове посилання (схема URL), наприклад:

```
<Script src = "https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js"> </script>
<A REF="JavaScript:JavaScript_код"> ... </A>
<IMG SRC = "JavaScript: JavaScript_код">
```

обробник подій (обробник), наприклад:

```
<FORM> <INPUT TYPE = button VALUE = "Кнопка" onClick = "window.alert ('intuit');"> </FORM>
```

підстановка (entity) (у Microsoft Internet Explorer реалізована у версіях від 5.X і вище), наприклад:

```
<FORM> <INPUT VALUE = "& {window.location.href};" SIZE = "& {l ()};" > </FORM>
```

1.1.3 Повідомлення про помилку

В основному бувають помилки двох типів: синтаксису і сценарію. Помилка синтаксису означає друкарську помилку або пропущений текст. Помилка сценарію означає, що можливо переплутали місцями команди або вставили неправильні.

Існують програми, які допомагають виправляти помилки, цей процес називається «**debugging**» («знищення багів, помилок.)

Рядок з помилкою потрібно відраховувати від самого верху документа HTML, а не від першого рядка JavaScript.

“Визначення відсутнє” – помилка сценарію; означає, що в скрипті щось не погоджено.

1.1.4 Коментарі

Виділяють однорядкові та багаторядкові коментарі (табл. 1.2).

Таблиця 1.2 – Види коментарів

Вид коментаря	Синтаксис
однорядковий	// однорядковий коментар
багаторядковий	/ * багатостроковий коментар * /

1.1.5 Методи, що дозволяють виводити діалогові вікна

В JavaScript реалізовано 3 методи, що дозволяють виводити користувачеві діалогові вікна, інформація щодо яких подана в табл. 1.3 та приклади продемонстровані на рис. 15-19.

Таблиця 1.3 – Методи, що дозволяють виводити діалогові вікна

Методи	**alert**	**confirm**	**prompt**
Використання	Використовується для виведення найпростішого діалогового вікна, що містить текст повідомлення та єдину кнопку "Ok".	Дозволяє вивести користувачеві діалогове вікно, що містить текст повідомлення і кнопки "Ok" і "Cancel"; використовується в тих випадках, коли користувач повинен зробити вибір.	Дозволяє вивести користувачеві діалогове вікно запиту на введення даних; використовується в тих випадках, коли користувач повинен ввести рядок тексту
Формат	`alert ("Текст повідомлення")`	`let result = confirm ("Текст питання") if (result) { / * Дії * / }`	`let result = prompt(title, [default])`
Особливості		Функція confirm повертає логічне значення в залежності від натиснутої користувачем кнопки: "Ok" відповідає значенню true, "Cancel" - значенням false. Як правило, результат роботи функції привласнюють змінній, для подальшого аналізу	Необхідно пам'ятати, що функція prompt повертає результат строкового типу. Тому, перш ніж його використовувати в арифметичних виразах, необхідно виконати перетворення типів до числового.

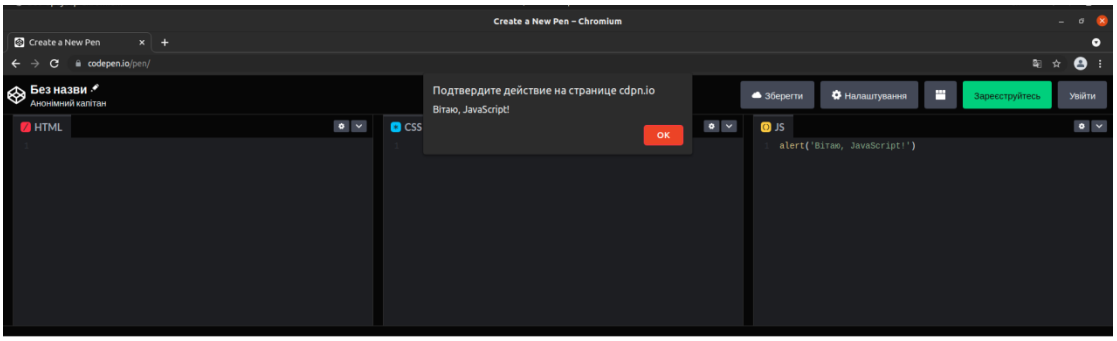

Рисунок 1.15 – Приклад використання методу alert в середовищі Codepen

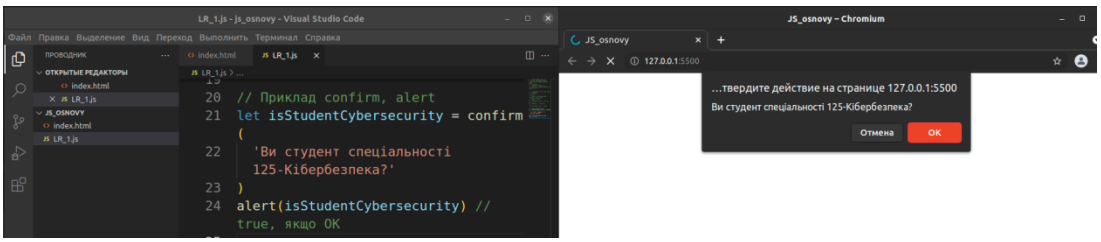

Рисунок 1.16 – Приклад використання методу confirm в VSCode

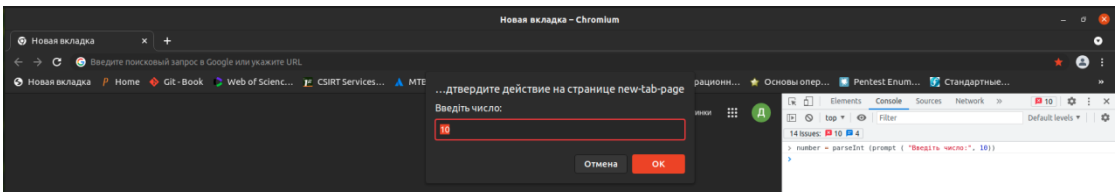

Рисунок 1.17 – Приклад використання методу prompt в консолі інструмента розробника

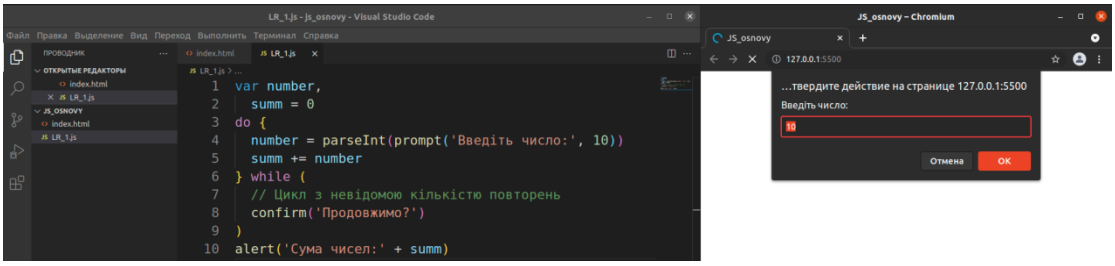

Рисунок 1.18 – Приклад з виведенням на сервер коду з використання методу prompt в VSCode

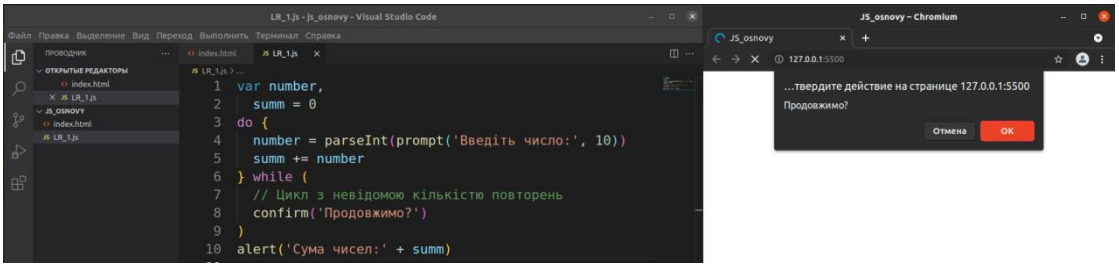

Рисунок 1.19 – Приклад з виведенням на сервер коду з використання методу confirm в VSCode

1.2 Завдання до лабораторної роботи

Під час виконання завдань фіксувати знімки екрану чи фото всіх кроків виконання для оформлення звіту, завдання пропонуються на базі середовища Visual Studio Code.

Завдання 1. Створення проекту, файлу .html.

Створити папку проекту, файл .html (рис. 1.20)

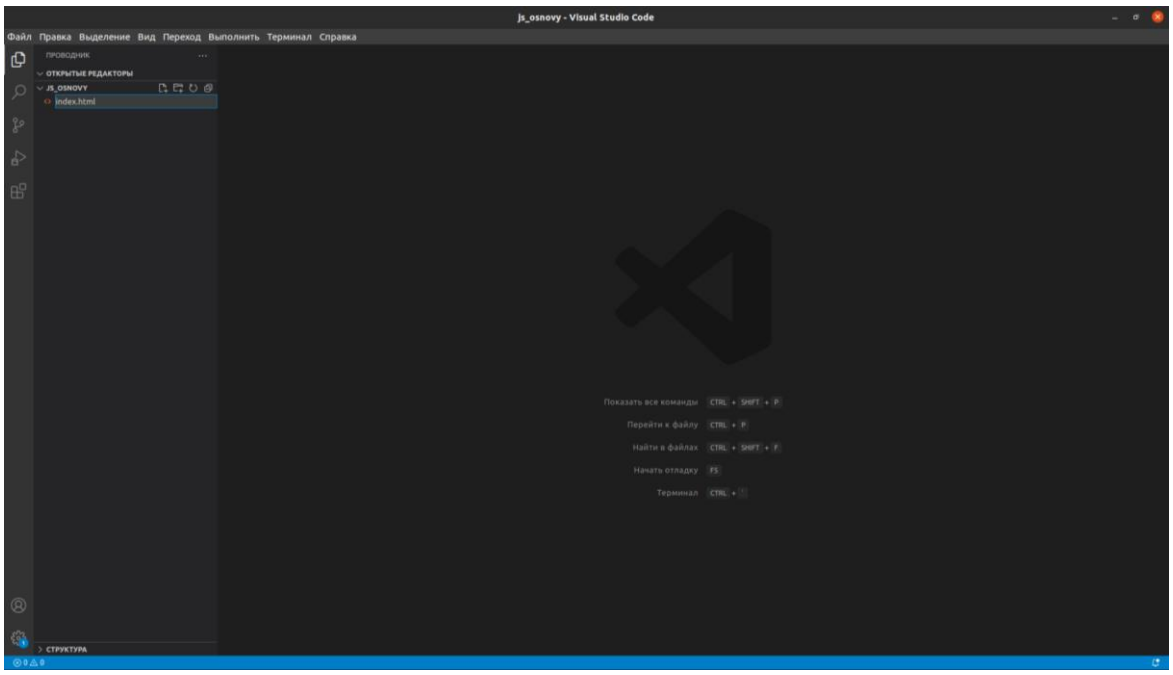

Рисунок 1.20 – Папка проекту, файл .html в VSCode

з початковим вмістом за замовчуванням (“!”, потім “Tab”), продемонстовано на рис. 1.21:

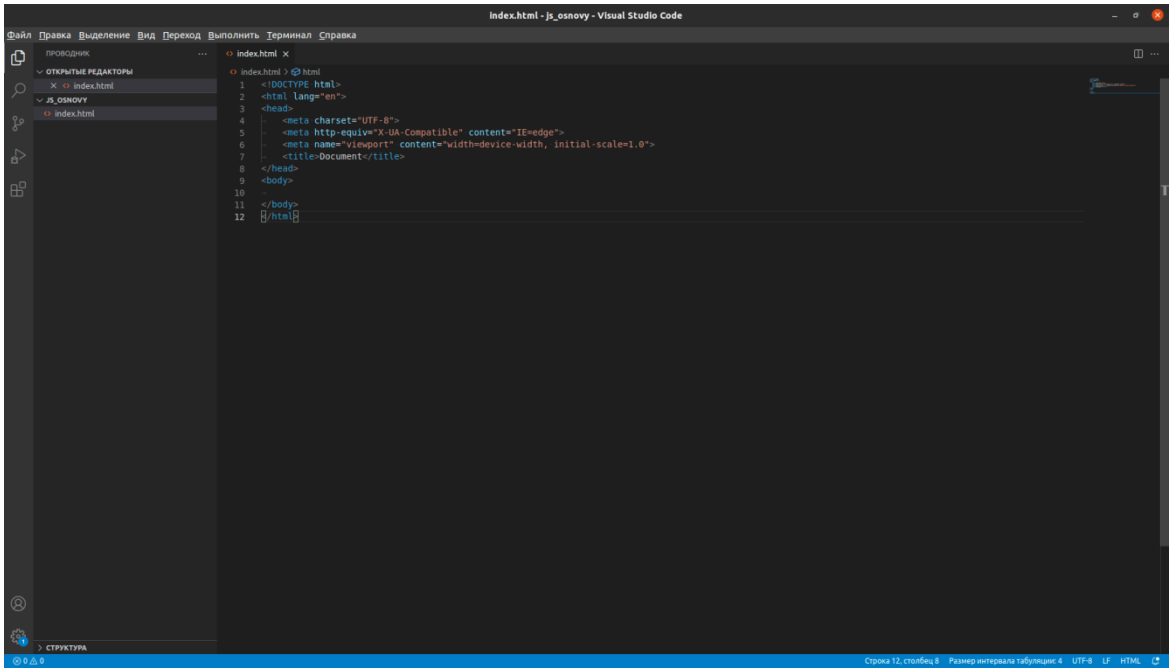

Рисунок 1.21 – Файл .html з вмістом за замовчуванням в VSCode

Внести зміни в файл .html (рис. 1.22):

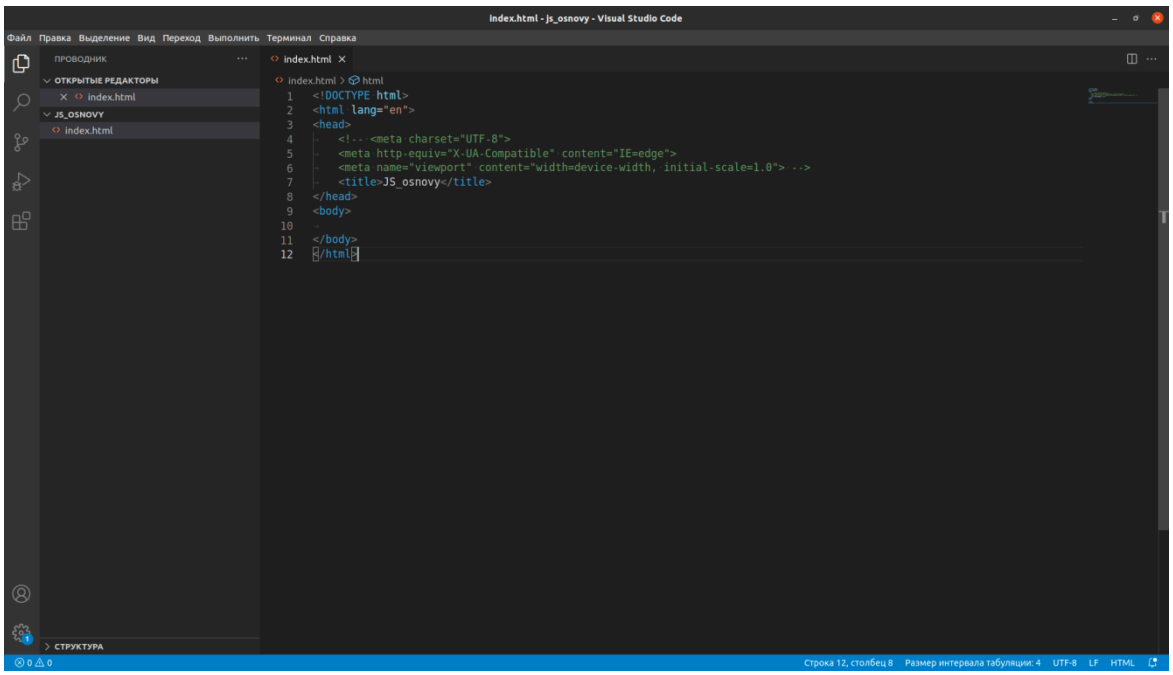

Рисунок 1.22 – Файл .html зі змінами в VSCode

Додатково: швидке відображення стартової сторінки (“!”, потім “Tab”) відбувається з використанням набору плагінів для текстових редакторів, які прискорюють написання коду HTML, XML, XSL, а також коду на деяких інших мовах – Emmet (рис.1.23): https://emmet.io/.

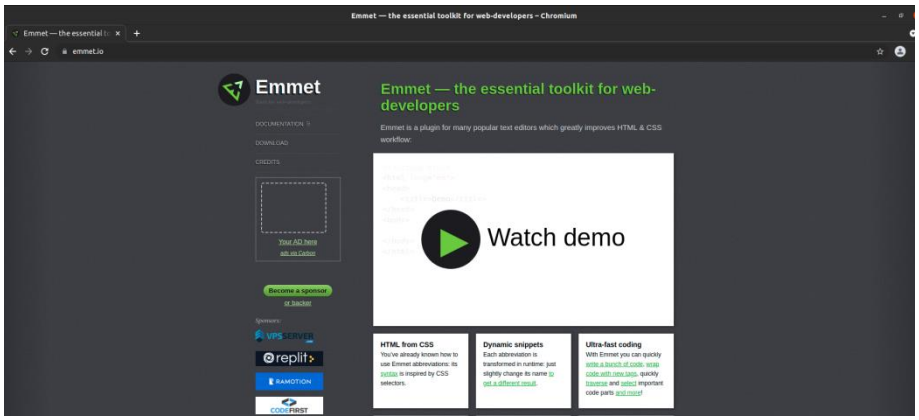

Рисунок 1.23 – Офіційна сторінка Emmet в VSCode

Завдання 2. Запуск сервера.

Варіанти запуску сервера:

1) через відкриття файлу .html в папці проекту (рис.1.24);

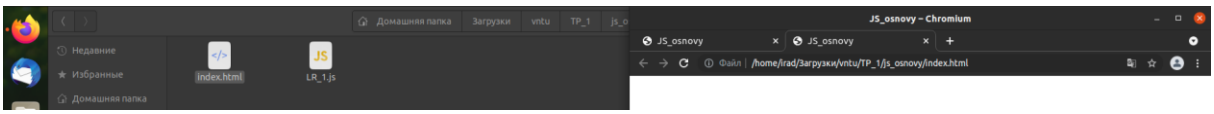

Рисунок 1.24 – Папка проекту з запуском сервера

2) для зручності роботи в VS Code можна встановити “Live Server”, процедуру встановлення та відкриття продемонстровано на рис. 1.25, 1.26;

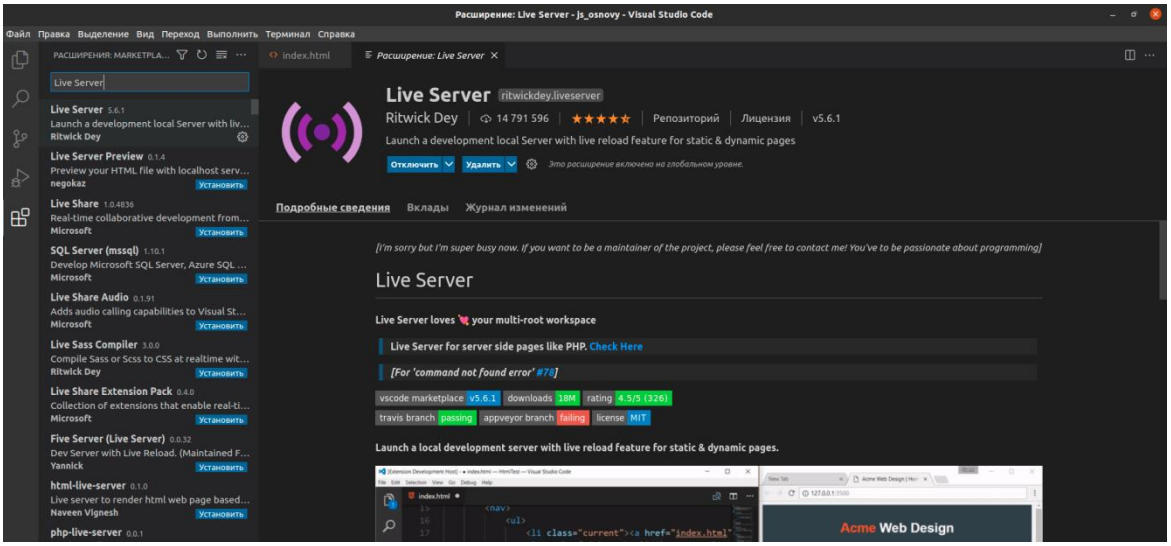

Рисунок 1.25 – Встановлення плагіну Live Server в VSCode

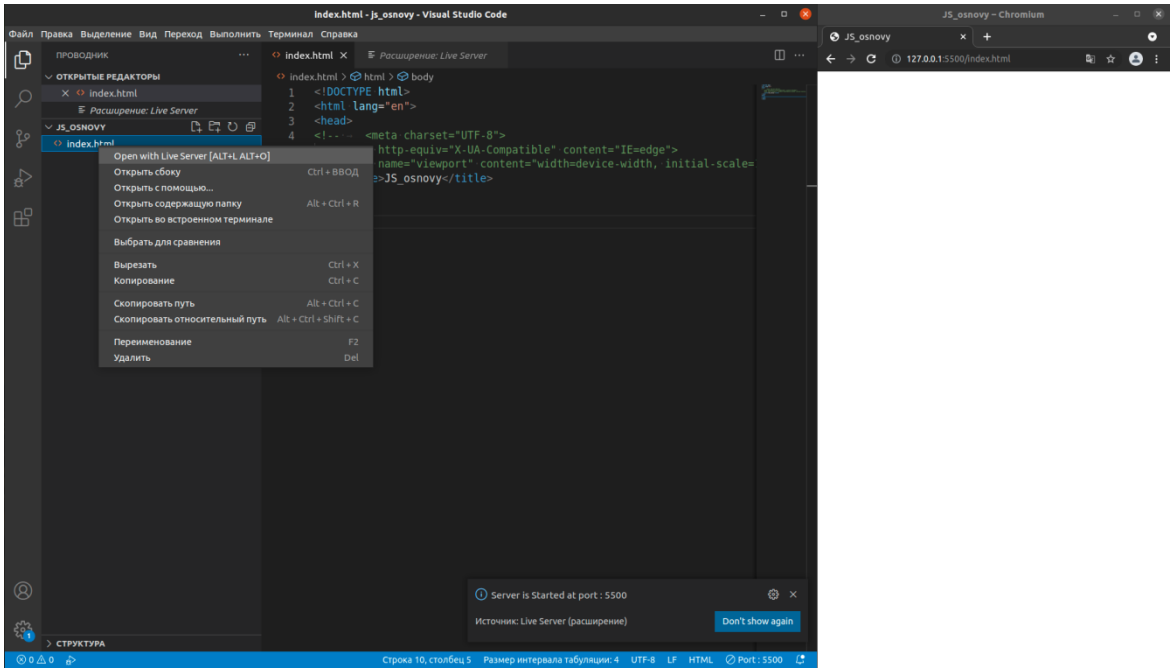

Рисунок 1.26 – Запуск локального сервера розробки з функцією перезавантаження для статичних та динамічних сторінок

Завдання 3. Виконання програмного коду в теговому контейнері <BODY> ... </BODY> HTML.

Протестувати (окремо або комплексно, з наповненням за вибором) методи, що дозволяють виводити користувачеві діалогові вікна: alert, confirm, prompt. Наприклад, рис. 1.27

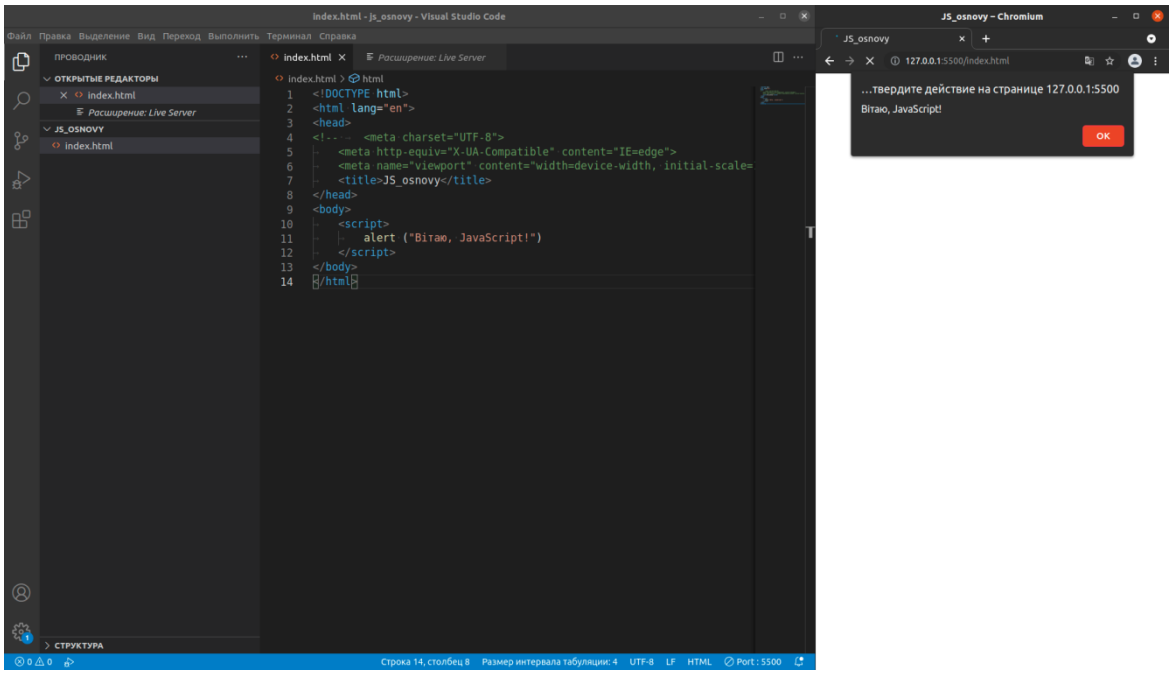

Рисунок 1.27 – Запуск локального сервера розробки з кодом

ЛАБОРАТОРНА РОБОТА № 2
ОБРАХУВАННЯ АРИФМЕТИЧНИХ ВИРАЗІВ. ВИКОРИСТАННЯ ВЛАСТИВОСТЕЙ І МЕТОДІВ MATH

Мета: ознайомитись з змінними, способами оголошення змінних, типами даних, базовими операторами, переліком властивостей і методів об'єкта Math.

2.1 Теоретичні відомості

2.1.1 Змінна

Змінна - це «іменоване сховище» для даних.

Виділяють кілька способів оголошення змінних для зберігання даних продемонстрованих в табл. 2.1.

Таблиця 2.1 – Способи оголошення змінної

Ключове слово	**var**	**let**	**const**
Спосіб оголошення	застарілий	сучасний	максимальне використання
Можливість зміни значення змінної	може змінюватися		не може змінюватися

Приклади	`var message`	`let message`
		або
		`let message = 'Hello!'`
		`//` визначаємо змінну і присвоюємо їй значення
			`const pi = 3,14`

Особливості імен	Обмеження: Ім'я змінної має містити тільки букви, цифри або символи $ і _. Перший символ не повинен бути цифрою. Особливості: Регістр має значення. Якщо ім'я містить кілька слів, зазвичай використовується верблюжа нотація, тобто, слова слідують одне за іншим, де кожне наступне слово починається з великої літери: myVeryLongName.		Особливості: Константи з іменами, записаними великими літерами, використовуються тільки як псевдоніми для «жорстко закодованих» значень.

2.1.2 Типи даних, перетворення типів

В JavaScript є 8 основних типів:

- **number** для будь-яких чисел: цілочисельних або чисел з плаваючою точкою; цілочисельні значення обмежені діапазоном ± (253-1);
- **bigint** для цілих чисел довільної довжини;
- **string** для рядків (рядок може містити нуль або більше символів, немає окремого символьного типу);
- **boolean** для true / false;
- **null** для невідомих значень – окремий тип, який має одне значення null (спеціальне значення, яке представляє собою «нічого», «порожньо» або «значення невідомо»; не є «посиланням на неіснуючий об'єкт» або «нульовим покажчиком», як в деяких інших мовах);
- **undefined** для неприсвоєння значень – окремий тип, який має одне значення undefined «Значення не було присвоєно»;
- **object** для більш складних структур даних;
- **symbol** для унікальних ідентифікаторів.

Оператор **typeof** дозволяє нам побачити, який тип даних збережений в змінній. Має дві форми: typeof x або typeof (x). Повертає рядок з ім'ям типу. Наприклад, "string". Для null повертається "object" – це помилка в мові, насправді це не об'єкт.

Існує 3 найбільш широко використовуваних **перетворення типів**:

- **строкове** відбувається, коли нам потрібно щось вивести (може бути викликане за допомогою String (value));
- **чисельне** – в математичних операціях (може бути викликане за допомогою Number (value), в таблиці 2.2 продемонстровані правила даних перетворень);

Таблиця 2.2 – Правила чисельних перетворень

Значення	Перетворення
undefined	NaN
null	0
true / false	1 / 0
string	Пробільні символи по краях обрізаються. Далі, якщо залишається порожній рядок, то отримуємо 0, інакше з непорожній рядки «зчитується» число. При помилці результат NaN.

– **логічне** – в логічних операціях (може бути викликане за допомогою Boolean (value); в таблиці 2.3 продемонстровані правила даних перетворень).

Таблиця 2.3 – Правила логічних перетворень

Значення	Перетворення
0, null, undefined, NaN, ""	false
будь-яке інше значення	true

Особливі випадки, в яких часто припускаються помилок:
undefined при чисельному перетворенні приймає значення NaN, а не 0;
"0" і рядки з одних пробілів типу "" при логічному перетворенні завжди приймають значення true.

2.1.3 Базові оператори

Операнд – значення, до якого застосовується оператор. Наприклад, в множенні 5 * 2 є два операнда: лівий операнд дорівнює 5, а правий операнд дорівнює 2. Іноді їх називають «аргументами» замість «операндів».

Унарним називається оператор, який застосовується до одного операнду.

Бінарним називається оператор, який застосовується до двох операндам.

2.1.3.1 Пріоритет операторів

Таблицю пріоритетів (зверніть увагу, що пріоритет унарних операторів вище, ніж відповідних бінарних), можна переглянути за посиланням:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

частина таблиці продемонстрована на рис. 2.1.

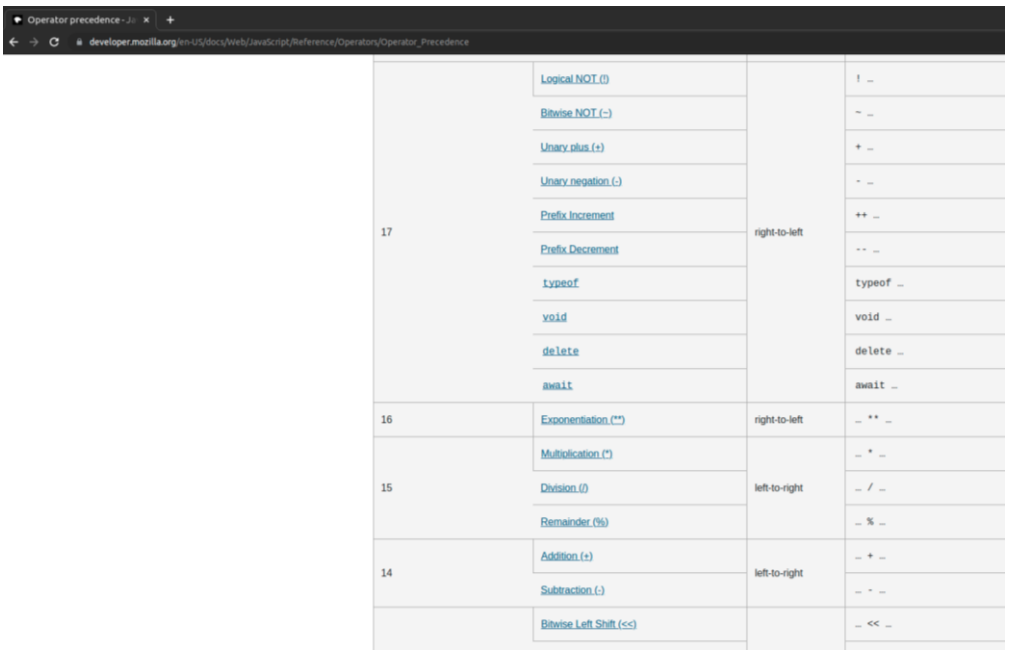

Рисунок 2.1 – Частина таблиці пріоритетів

Кожен оператор має відповідний номер пріоритету. Оператор, який має більший номер пріоритету, виконується раніше. Якщо пріоритет однаковий, то порядок виконання - зліва направо.

2.1.3.2 Оператор присвоєння

В таблиці пріоритетів є оператор присвоєння «=», у якого один з найнижчих пріоритетів: 3. Саме тому, коли змінній щось привласнюють, наприклад, `x = 2 * 2 + 1`, то спочатку виконається арифметика, а вже потім відбудеться присвоєння = зі збереженням результату в *x*.

2.1.3.3 Математичні оператори

Підтримуються наступні **математичні оператори**:

«+» – додавання;

«-» – віднімання;

«*» – множення;

«/» – ділення;

«%» – взяття залишку від ділення;

«**» – піднесення до степеня.

2.1.3.4 Інкремент / декремент

Однією з найбільш поширених числових операцій є збільшення або зменшення на одиницю. Для цього існують спеціальні оператори:

інкремент «++», який збільшує змінну на 1, наприклад:

```
let counter = 2;
counter ++; // працює як counter = counter + 1alert (counter); // 3
```

декремент «--», який зменшує змінну на 1:

```
let counter = 2;
counter--; // працює як counter = counter – 1
alert (counter); // 1
```

2.1.3.5 Побітові оператори

Побітові оператори працюють з 32-розрядними цілими числами (при необхідності приводять до них), на рівні їх внутрішнього двійкового представлення.

Підтримуються наступні побітові оператори:

- AND (і) (&);
- OR (або) (|);
- XOR (побітовое виключає або) (^);
- NOT (не) (~);
- LEFT SHIFT (лівий зсув) (<<);
- RIGHT SHIFT (правий зсув) (>>);
- ZERO-FILL RIGHT SHIFT (правий зсув із заповненням нулями) (>>>).

2.1.3.6 Оператор «кома»

Оператор «кома» (,) надає можливість обчислювати кілька виразів, розділяючи їх комою.

Кожен вираз виконується, але повертається результат тільки останнього. Наприклад:

```
let a = (1 + 2, 3 + 4);
alert (a); // 7 (результат обчислення 3 + 4)
```

Перший вираз 1 + 2 виконується, а результат відкидається. Потім йде 3 + 4, вираз виконується і повертається результат.

2.1.3.7 Оператори порівняння

В JavaScript виділяють такі оператори порівняння:

– більше / менше: a> b, a <b;
– більше / менше або дорівнює: a> = b, a <= b;
– рівність: a == b (зверніть увагу, для порівняння використовується подвійний знак рівності «==»);
– не дорівнює (в математиці позначається символом ≠, але в JavaScript записується як a! = B).

Оператори порівняння повертають значення логічного типу.

Рядки порівнюються посимвольно в лексикографічному порядку.

Значення різних типів при порівнянні приводяться до числа. Винятком є порівняння за допомогою операторів суворої рівності / нерівності.

Значення null і undefined рівні == один одному і не рівні будь-якому іншому значенню.

2.1.4 Math

Об'єкт **Math** є вбудованим об'єктом, що зберігає в своїх властивостях і методах різні математичні константи і функції.

Об'єкт Math не є функціональним об'єктом. На відміну від інших глобальних об'єктів, об'єкт Math не є конструктором. Всі властивості і методи об'єкта Math є статичними.

Math не працює з числами типу BigInt.

Перелік властивостей і методів можна переглянути за посиланням: https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Math.

2.2 Завдання до лабораторної роботи

Завдання 1. Підключення скрипта лабораторної роботи до HTML з застосуванням коментарів.

Створення файлу скрипта (рис. 2.2)

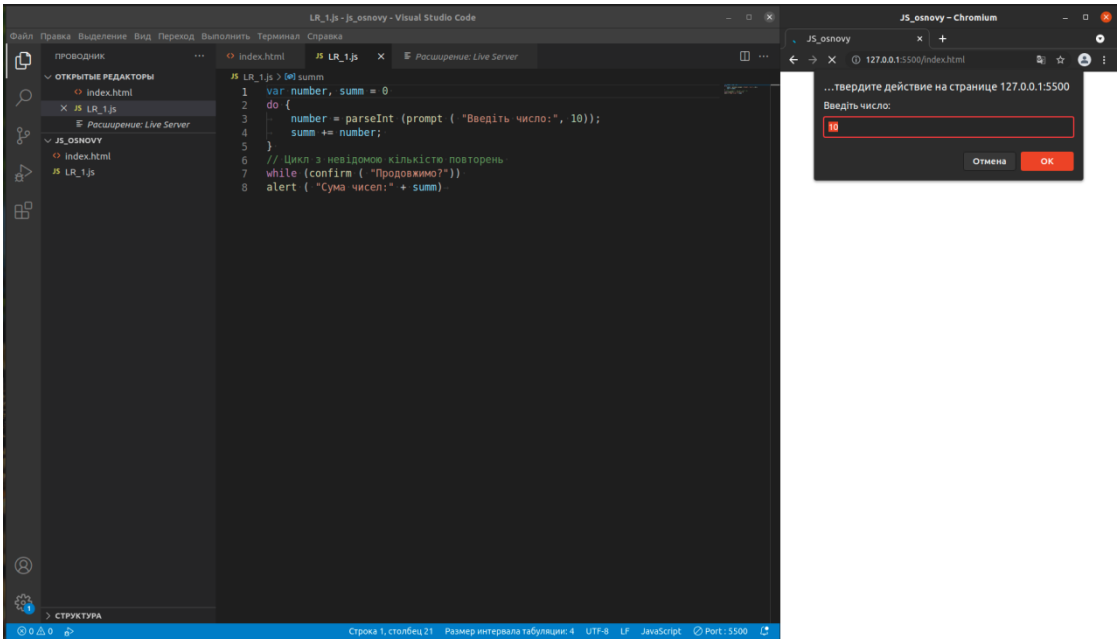

Рисунок 2.2 – Файл скрипта з індивідуальним завданням

Підключення скрипта до HTML, продемонстровано на рис. 2.3.

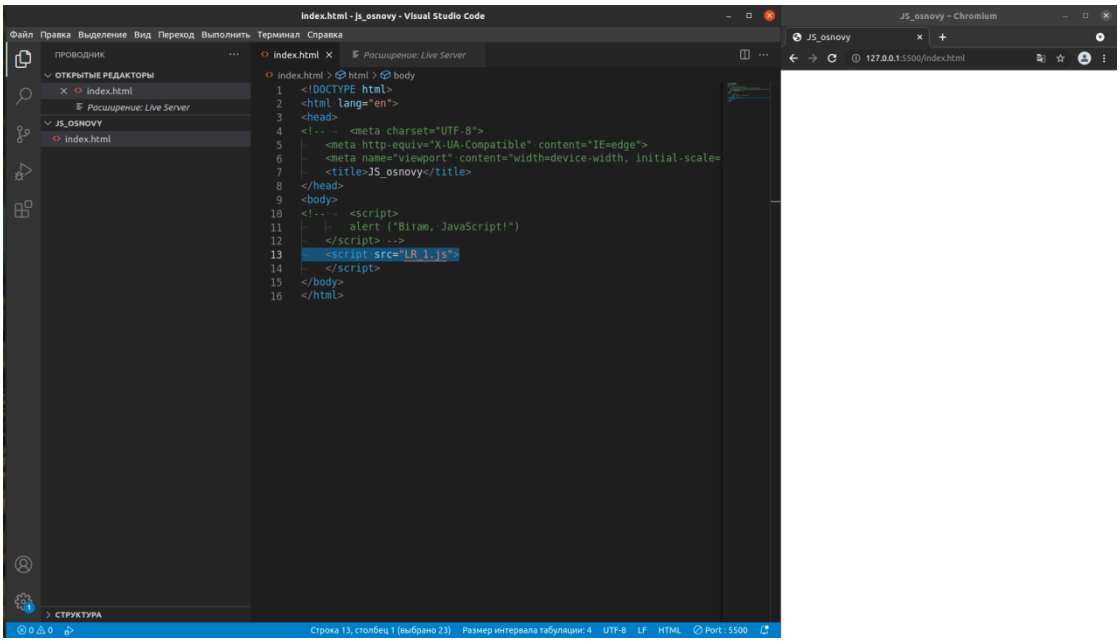

Рисунок 2.3 – Підключення скрипта до HTML

Завдання 2. Написати код відповідно до варіанту (див. Додаток А) в якому:

1) отримати інформацію від користувача щодо значень змінних у варіантах (використовуючи методи, що дозволяють виводити користувачеві діалогові вікна);

2) провести розрахунки відповідно до варіанту (застосувати також можливості об'єкту Math);

3) вивести результат;

4) додатково: врахувати умову при яких змінних вираз немає змісту.

ЛАБОРАТОРНА РОБОТА № 3
УМОВНІ КОНСТРУКЦІЇ.
ОПЕРАТОРИ ПОРІВНЯННЯ, ЛОГІЧНІ ОПЕРАТОРИ

Мета: ознайомитись умовними конструкціями, операторами порівняння, логічними операторами.

3.1 Теоретичні відомості

3.1.1 Умовні конструкції

3.1.1.1 Оператор if … else

В таблиці 3.1 продемонстровані варіації синтаксису оператора **if ... else,** на рис. 3.1-3.11 наведені приклади використання.

Таблиця 3.1 – Варіації синтаксису оператора if ... else

Варіації оператора	Синтаксис, використання	Особливості синтаксису

базовий if ... else	`if (condition) {`
	`  code to run if condition is true`
	`} else {`
	`  run some other code instead`
	`}`
		ключове слово if; умова для перевірки (condition); в дужках {} код, який буде виконуватися, коли умова істина (true); ключове слово else; в {} код, який виконається, якщо умова не істина (НЕ true).

скорочена форма if ... else	`if (condition) code to run if condition is true` `else run some other code instead`	без використання дужок {}

if	`if (condition) {`
	`  code to execute if condition is true`
	`}`
	`some other code`
		else і другий блок дужок {} не обов'язковий

else if	спосіб прив'язати додаткові варіанти / результати до if ... Else, використовуючи else if	для кожного додаткового вибору є потреба у додатковому блоці, який між if () {...} і else {...}
вкладений if ... else	використання одного умовного оператора if ... else всередині іншого, тобто вкладення	
Тернарний оператор (умовний оператор "?")	`(умова)? виконати цей код: виконати цей код замість першого`	альтернатива блоку if ... else, яка дозволяє скоротити код: два варіанти в одну стрічку, які обираються на основі умови

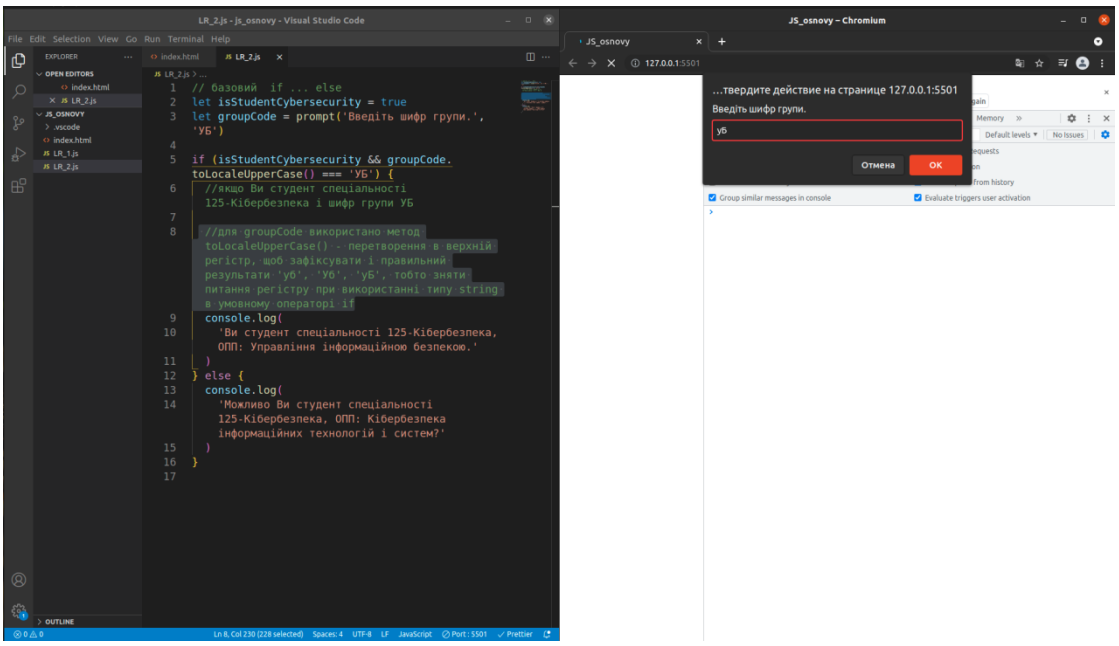

Рисунок 3.1 – Приклад для базового if ... else: запит на введення інформації користувачем

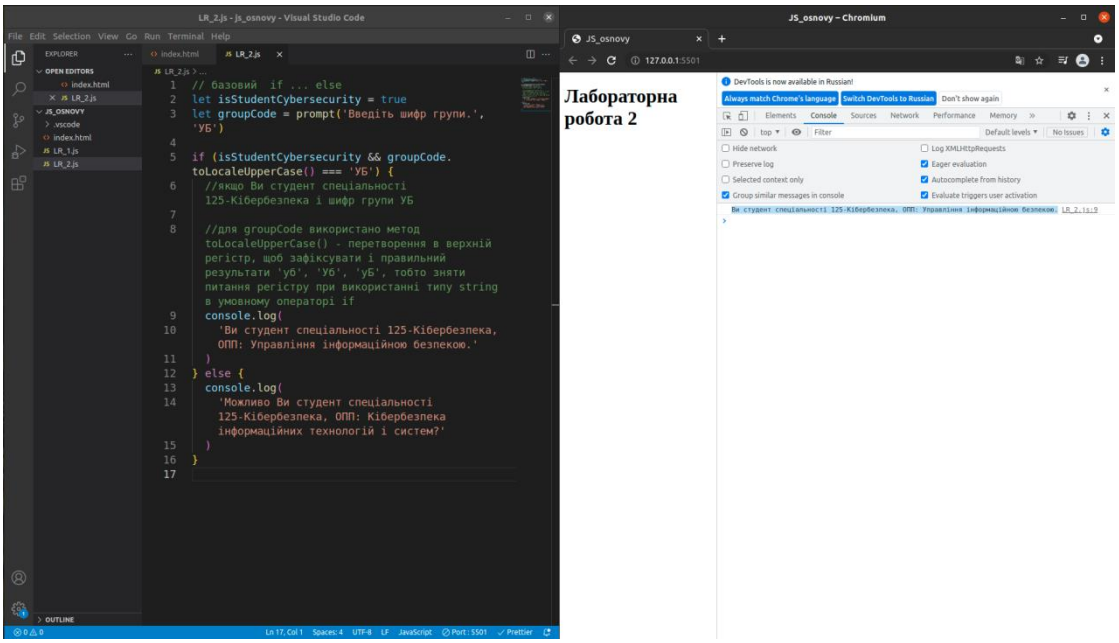

Рисунок 3.2 – Приклад для базового if ... else: результат відповідно до введеної інформації

Додатково: при роботі з умовою в умовному операторі if , де використовуються string-овий тип даних, використовувати приведення до певного регістру (нижнього, верхнього) за допомогою методів:

toLowerCase() – повертає значення стрічки, на якій він був викликаний, у нижній регістр; детальніше за посиланням:
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/toLowerCase

toLocaleUpperCase() – овертає значення стрічки, на якій він був викликаний, у верхній регістр; детальніше за посиланням: https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/toUpperCase

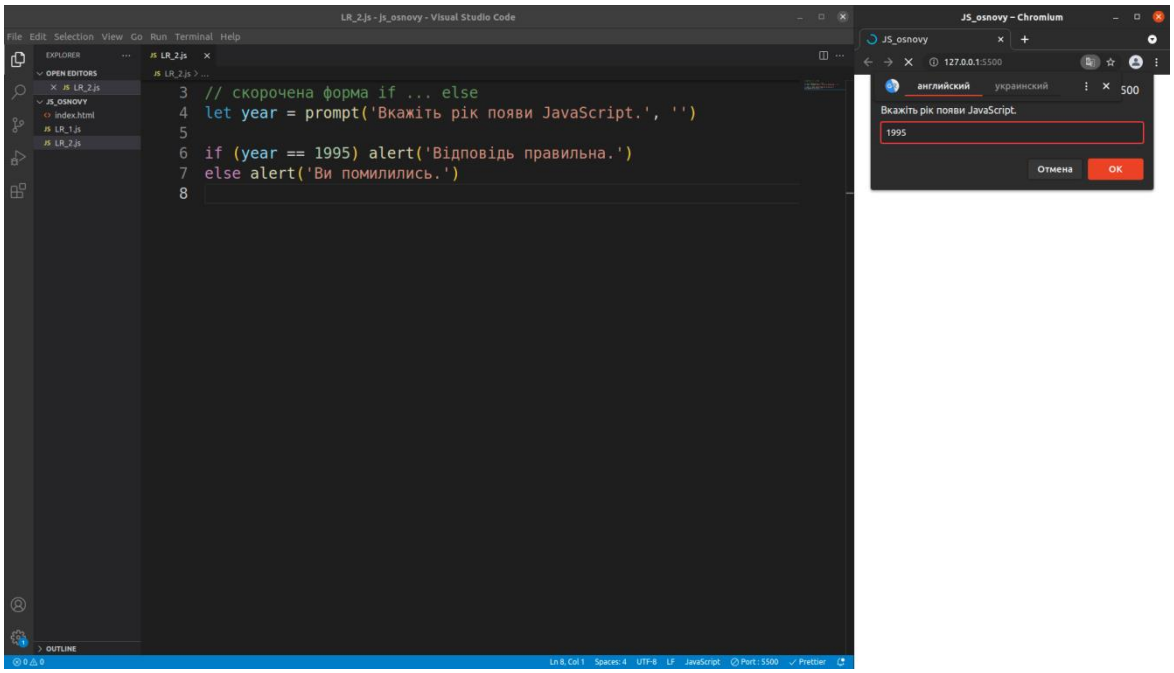

Рисунок 3.3 – Приклад використання скороченої форми if ... else: запит на введення інформації користувачем

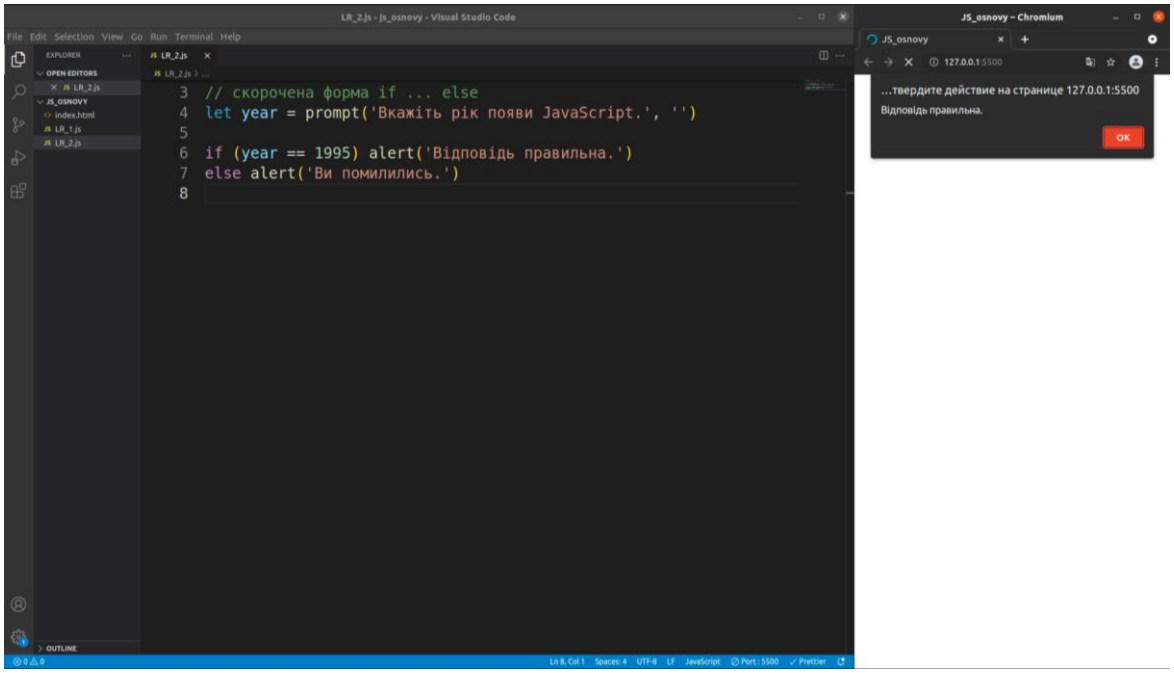

Рисунок 3.4 – Приклад використання скороченої форми if ... else: результат відповідно до введеної інформації

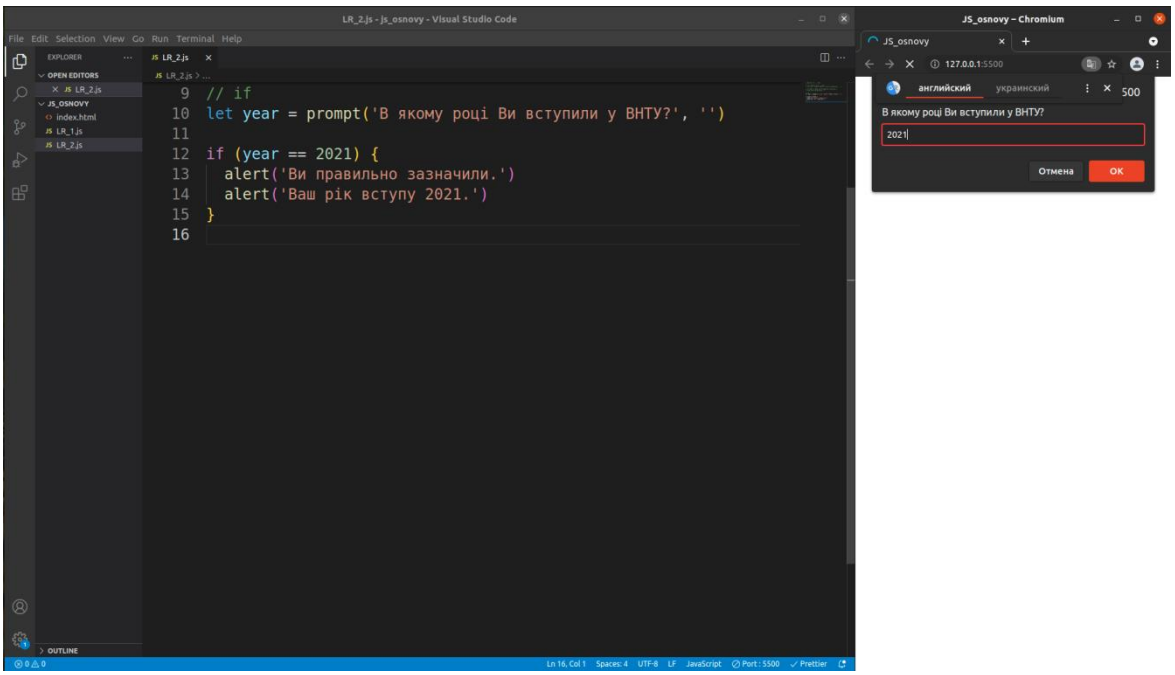

Рисунок 3.5 – Приклад використання if: запит на введення інформації користувачем

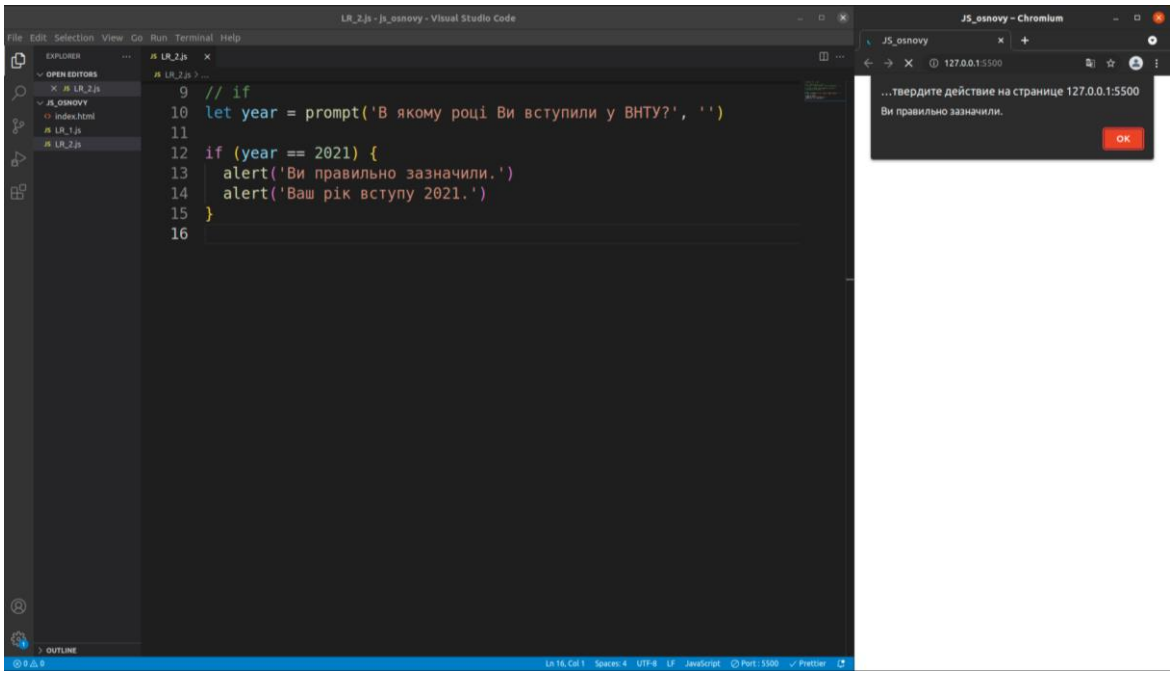

Рисунок 3.6 – Приклад використання if: виведення першого повідомлення

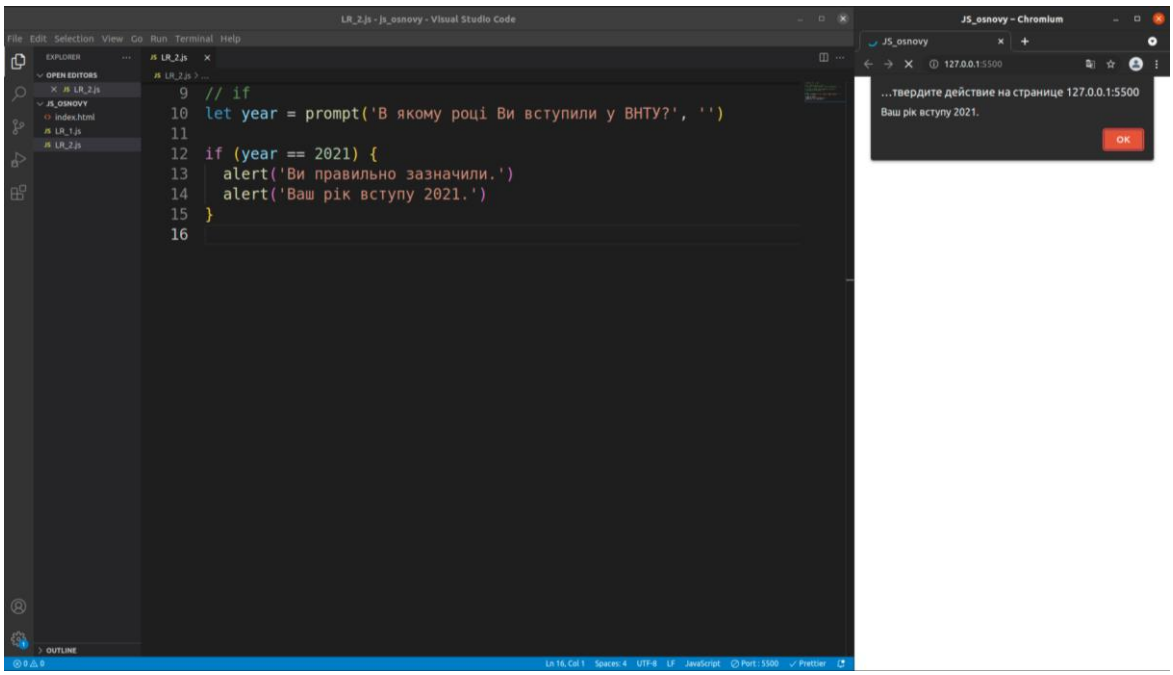

Рисунок 3.7 – Приклад використання if: виведення другого повідомдення

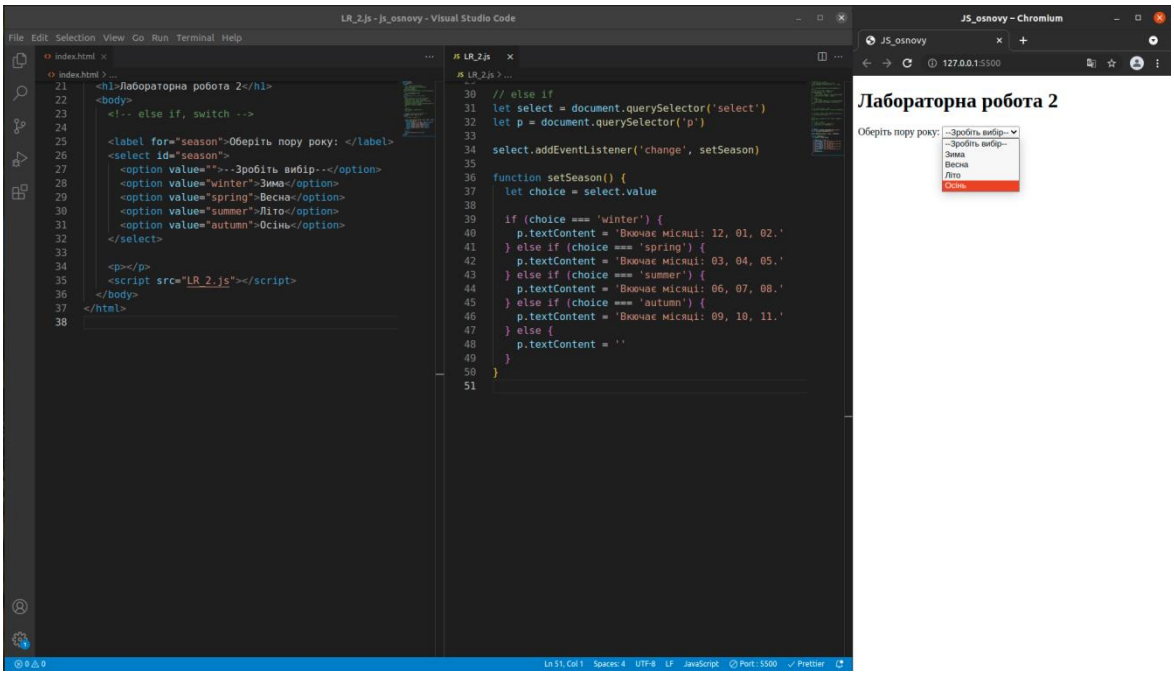

Рисунок 3.8 – Приклад використання else if: запит на вибір у селекторі інформації користувачем

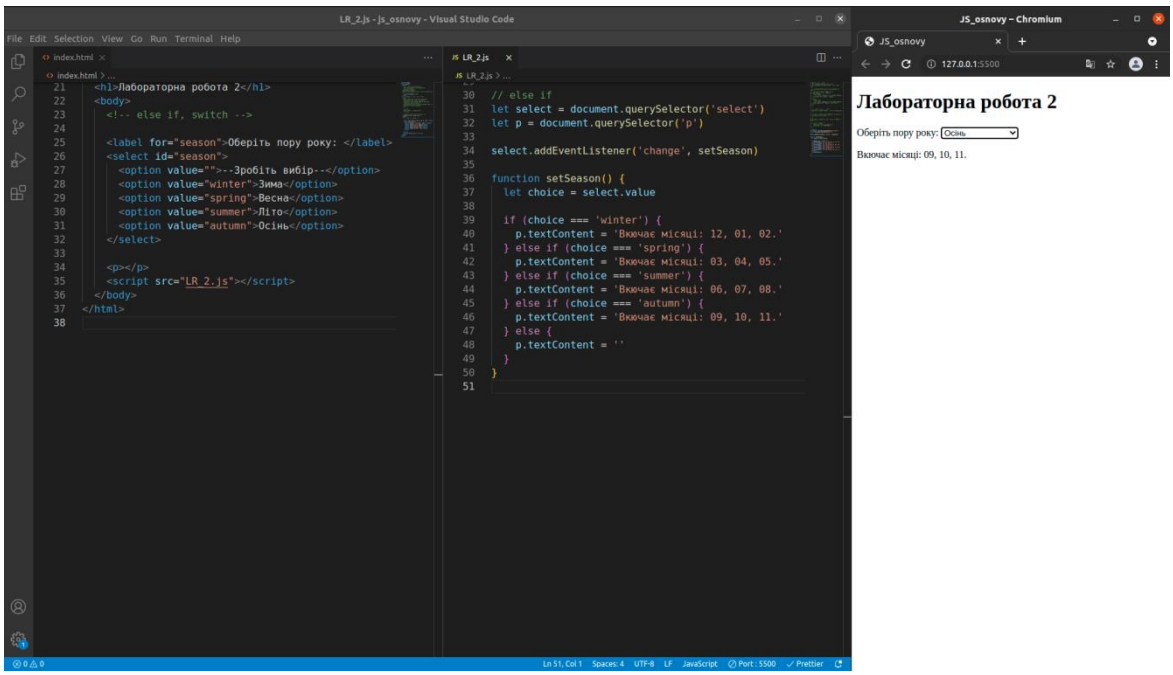

Рисунок 3.9 – Приклад використання else if: результат вибору

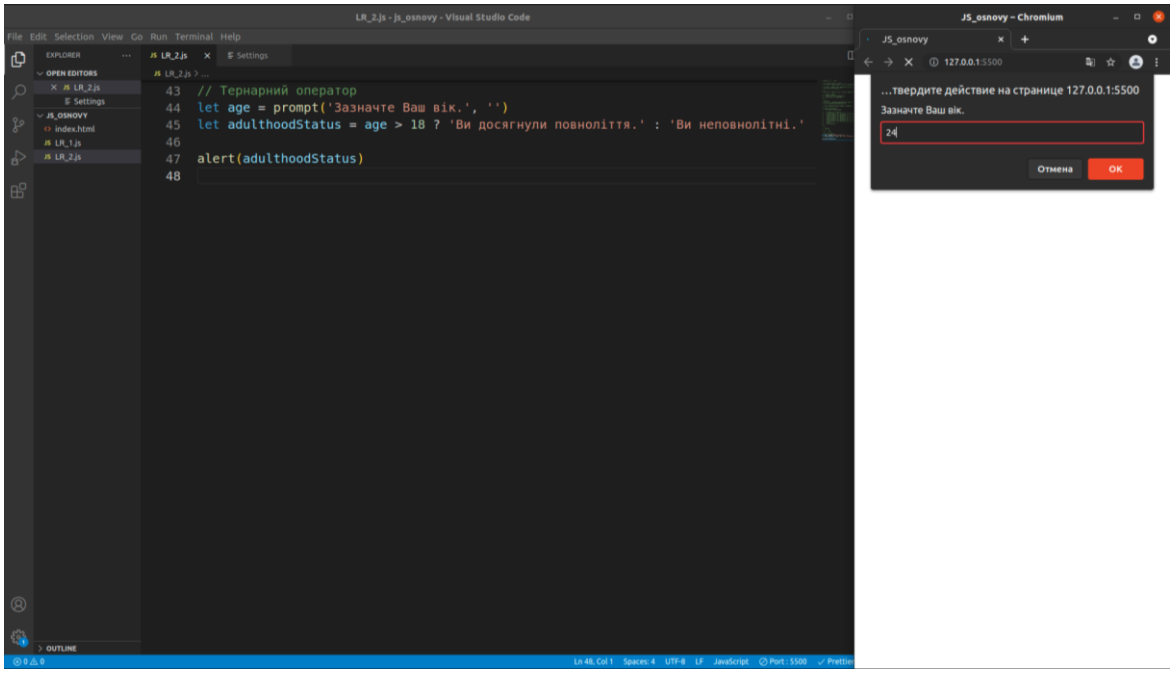

Рисунок 3.10 – Приклад використання тернарного оператора: запит на введення інформації користувачем

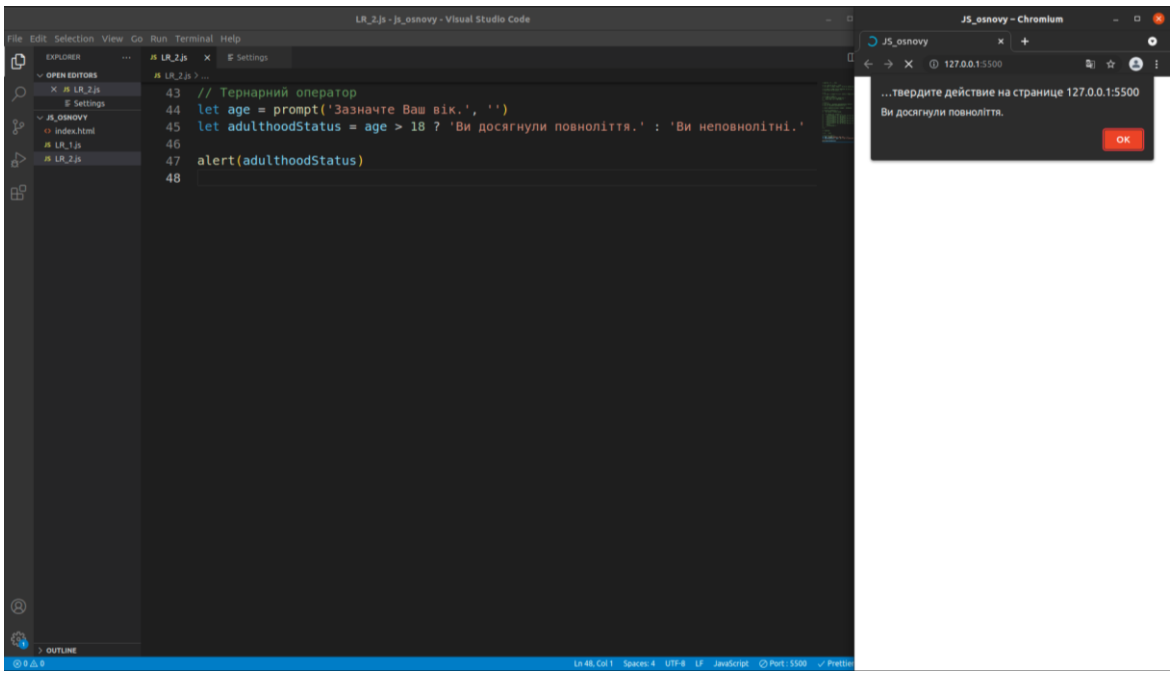

Рисунок 3.11 – Приклад використання тернарного оператора: результат відповідно до введеної інформації

3.1.1.2 Оператор switch

Оператор **switch** – приймає однин вираз або значення, а потім переглядає ряд варіантів, поки не знайде варіант, що відповідає цьому значенню, після чого виконує код, призначений цим варіантом (див. табл. 3.2 та приклад на рис. 3.12, 3.13).

Таблиця 3.2 – Особливості синтаксису switch

Синтаксис	Особливості синтаксису

`switch (вираз) {`
`  case choice1:`
`  виконати даний код`
`  break;`
`  case choice2:`
`  виконати даний код, а не попередній`
`  break;`
 	
`// варіантів може бути певна кількість`
 	
`  default:`
`  а взагалі-то, виконати тільки даний код`
`}`
	ключове слово switch, за яким пара дужок {};
	у дужках наводиться вираз або значення;
	ключове слово case, за яким варіант вибору перевіряється на відповідність виразу або значенням і двокрапка;
	код, для виконаня, якщо варіант збігається з виразом;
	оператор break, за яким слідує крапка з комою;
	якщо варіант співпав з виразом або значенням, браузер закінчить виконувати блок коду, дійшовши до break, і перейде до виконання коду, розташованого після switch;
	варіантів вибору може бути будь-яка кількість;
	ключове слово default використовується так, як будь-який інший варіант вибору за тим винятком, що після default немає інших варіантів вибору, тому інструкція break не потрібна, ніякого коду далі немає (це варіант вибору за замовчуванням, який обирається, якщо жоден з варіантів не співпав з виразом).

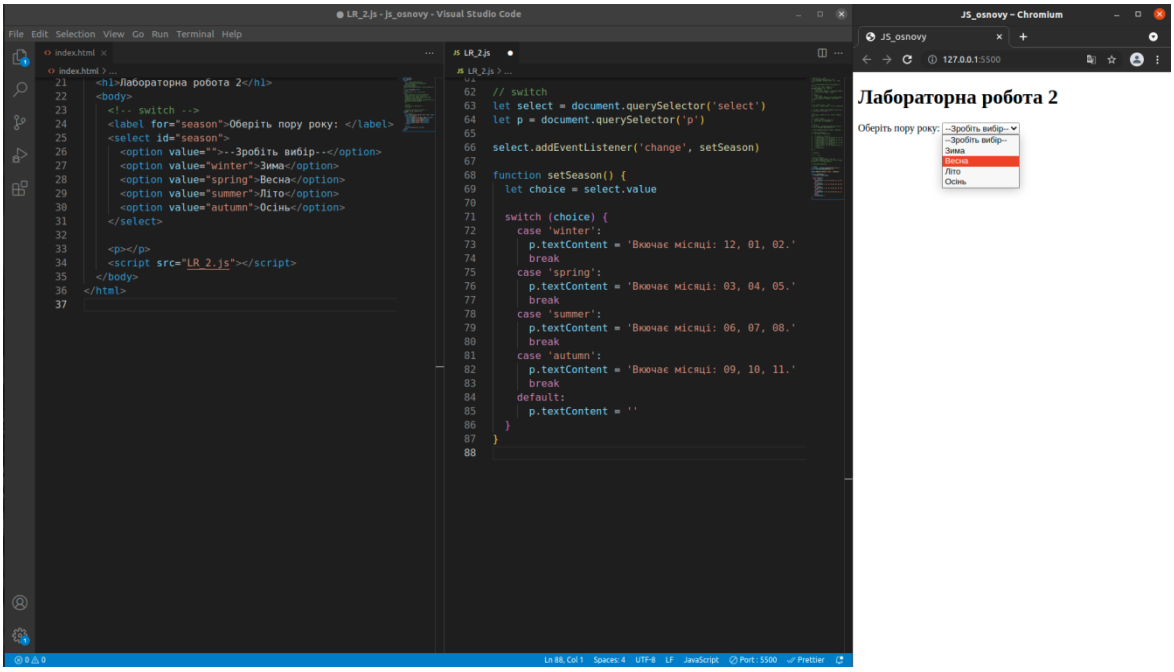

Рисунок 3.12 – Приклад використання switch: запит на вибір у селекторі інформації користувачем

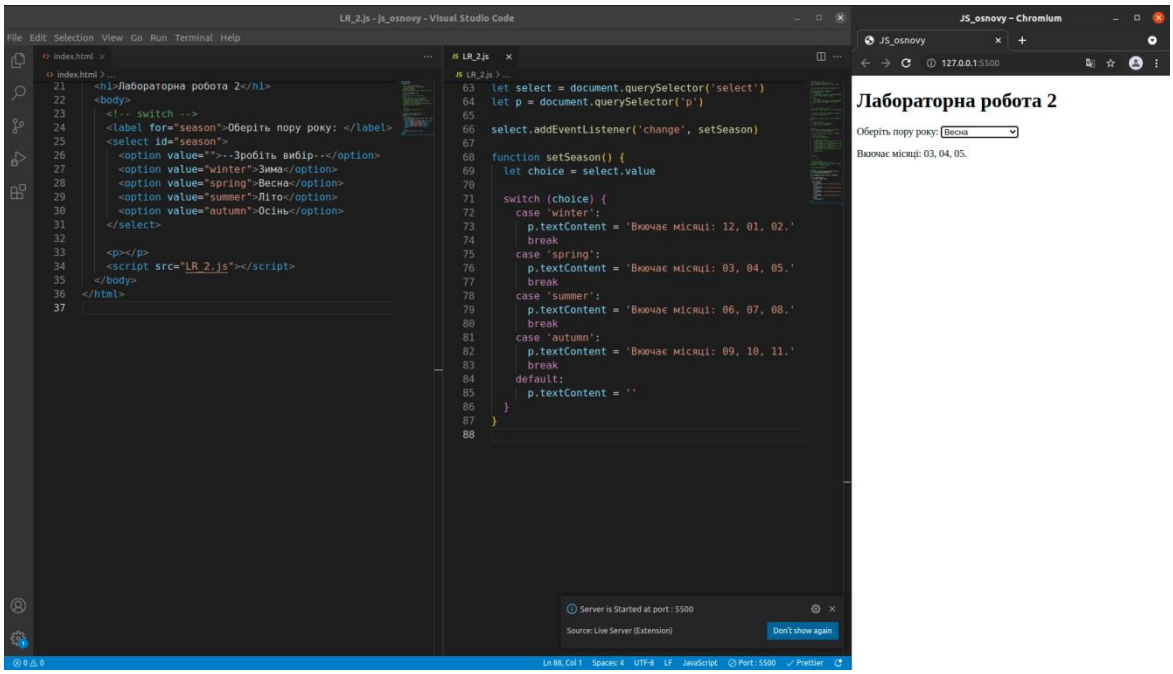

Рисунок 3.13 – Приклад використання switch: результат вибору

3.1.2 Оператори порівняння

В JavaScript виділяють ряд операторів порівняння, які продемонстровані в табл. 3.3.

Таблиця 3.3 – Оператори порівняння

Оператор	Назва	Призначення
==	рівність	перевіряє на рівність значень, не розглядаючи їх типи
===	сувора рівність	перевіряє ліве і праве значення на ідентичність (перевіряє типи операндів)
!=	нерівність	перевіряє на нерівність значень, не розглядаючи їх типи
! ==	сувора нерівність	перевіряє ліве і праве значення на неідентичність (перевіряє типи операндів)
<	менше	перевіряє, чи менше ліве значення правого
>	більше	перевіряє, чи більше ліве значення правого
<=	менше або дорівнює	перевіряє, чи менше ліве значення правому (або дорівнює йому)
> =	більше або дорівнює	перевіряє, більше чи ліве значення лівого (або дорівнює йому)

Зауваження: Число 0, порожній рядок "", null, undefined і NaN приймають значення false. Через це їх називають «помилковими» («falsy») значеннями. Решта значенньприймають значення true, тому їх називають «правдивими» («truthy»).

3.1.3 Логічні оператори

В JavaScript виділяють такий перелік логічних операторів: *І, АБО і НЕ*, призначення яких описане в табл. 3.4.

Таблиця 3.4 – Логічні оператори

Оператор	Назва	Призначення
&&	І	дозволяє об'єднати два або більше вирази так, що кожне з них окремо має мати значення true, щоб в результаті загальне вираз мало значення true
||	АБО	дозволяє об'єднати два або більше вирази так, що одне або декілька з них повинна мати значення true, щоб в результаті загальне вираз мало значення true
!	НЕ	буде заперечувати

3.1.4 Пошук: getElement, querySelector**

Document Object Model, скорочено DOM - об'єктна модель документа, яка представляє весь вміст сторінки у вигляді об'єктів, які можна змінювати. Об'єкт document - основна «вхідна точка».

Виділяють 6 основних методів пошуку елементів в DOM (табл. 3.5)

Таблиця 3.5 – Основні методи пошуку елементів в DOM

Метод	Пошук
querySelector	CSS-selector
querySelectorAll	CSS-selector
getElementById	id
getElementsByName	name
getElementsByTagName	tag or '*'
getElementsByClassName	class

3.1.5 Обробник element.addEventListener

Подія - це сигнал від браузера про те, що щось сталося. Всі DOM-вузли подають такі сигнали (хоча події бувають і не тільки в DOM).

Подіям можна призначити **обробник**, тобто функцію, яка спрацює, як тільки подія відбулася. Саме завдяки обробникам JavaScript-код може реагувати на дії користувача.

Є три способи призначення обробників подій:

– атрибут HTML: onclick = "...";
– DOM-властивість: elem.onclick = function;
– спеціальні методи: elem.addEventListener (event, handler [, phase]) для додавання, removeEventListener для видалення.

Синтаксис додавання обробника:

```
element.addEventListener (event, handler [, options]);
```

event – ім'я події, наприклад "click";

handler – посилання на функцію-обробник;

options – додатковий об'єкт з властивостями:

once: якщо true, тоді обробник буде автоматично видалений після виконання;

capture: фаза, на якій повинен спрацювати обробник, історично склалося, що options може бути false / true, тобто {capture: false / true};

passive: якщо true, то обробник ніколи не викличе preventDefault ().

Для видалення обробника слід використовувати removeEventListener:

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

Видалення вимагає саме ту функцію-обробник яка була призначена.

3.2 Завдання до лабораторної роботи

Завдання 1. В проект підключити скрипт лабораторної роботи.

Завдання 2. Написати код виконання задачі, умови яких запропоновано поваріантно (див. Додаток Б). В коді:

1) використати умовні конструкції;

2) використати методи, що дозволяють виводити користувачеві діалогові вікна

3) за бажанням: використати: пошук: getElement*, querySelector* , обробник element.addEventListener.

ЛАБОРАТОРНА РОБОТА № 4
ЦИКЛИ.
ОПЕРАТОР BREAK. ІНСТРУКЦІЯ CONTINUE

Мета: ознайомитись з циклами, оператором break, інструкцією continue, з використанням міток.

4.1 Теоретичні відомості

4.1.1 Цикл while

В таблиці 4.1 продемонстровані варіанти синтаксису циклу **while**, на рис. 4.1, 4.2 наведений приклад використання.

Таблиця 4.1 – Варіанти синтаксису циклу while

Варіанти оператора	Синтаксис	Особливості синтаксису
while	`while (condition) { instruction }`	умова – вираз, логічне значення якого перевіряється щоразу перед входом в цикл, якщо значення істинне, то інструкція виконується, якщо хибне, то виконується код, наступний за циклом; інструкція – виконується, поки умова істинна; щоб виконати кілька інструкцій, використовують блоковий оператор ({...}) для їх групування.
скорочена формаwhile	`while (condition) instruction`	якщо тіло циклу складається з однієї інструкції, можна не використовувати фігурні дужки {...}
	`while (true)`	безкінечний цикл
	`while (i)`	короткий варіант while (i! = 0)

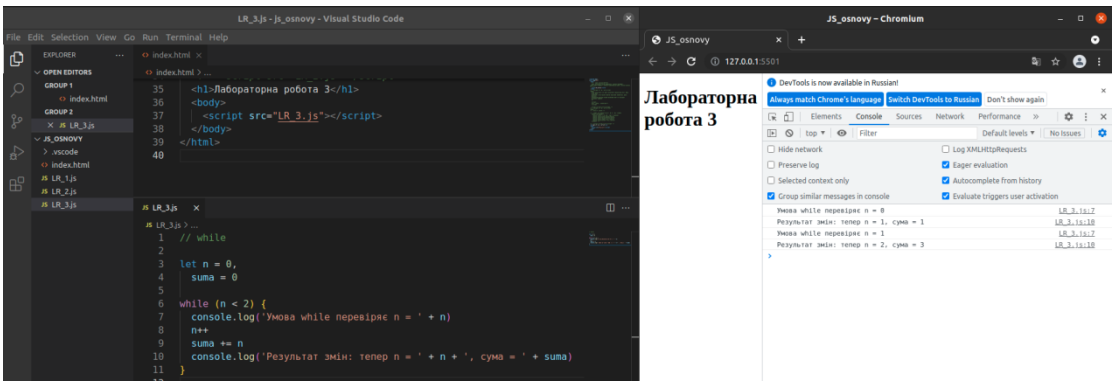

Рисунок 4.1 – Приклад використання while

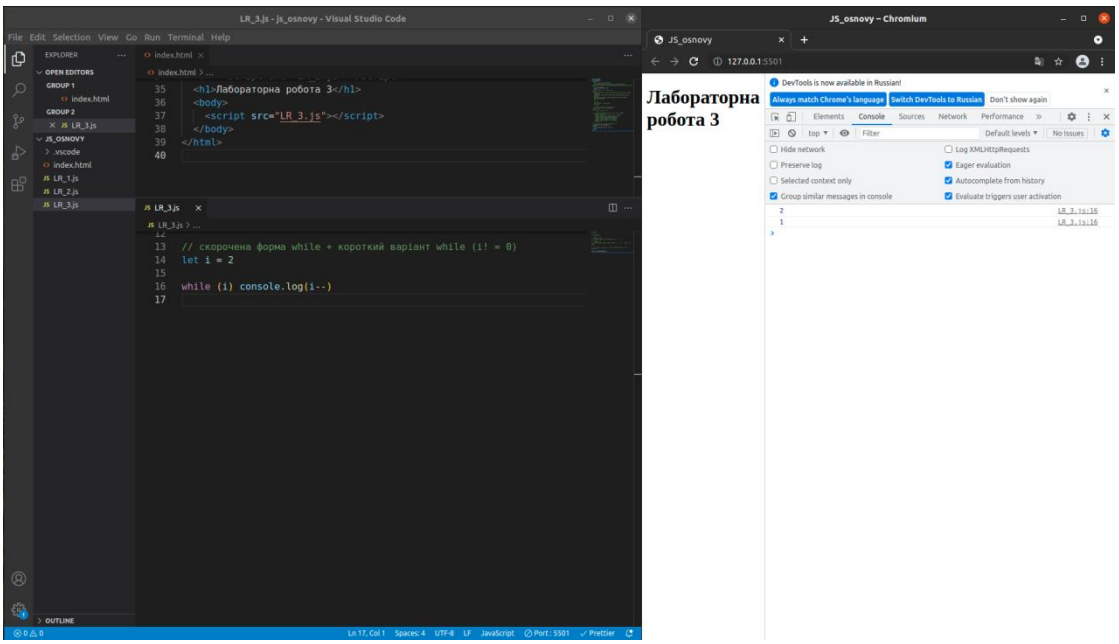

Рисунок 4.2 – Приклад використання скороченої форми while та короткого варіанту while (i! = 0)

4.1.2 Цикл do...while

В таблиці 4.2 продемонстровано особливості синтаксису циклу **do...while**, на рис. 4.3, 4.4 наведений приклад використання.

Таблиця 4.2 – Особливості синтаксису циклу do...while

Синтаксис	Особливості синтаксису
`do { // тіло цикла } while (condition)`	Цикл спочатку виконає тіло, а потім перевірить умову (condition), і поки її значення дорівнює true, він буде виконуватися знову і знову.

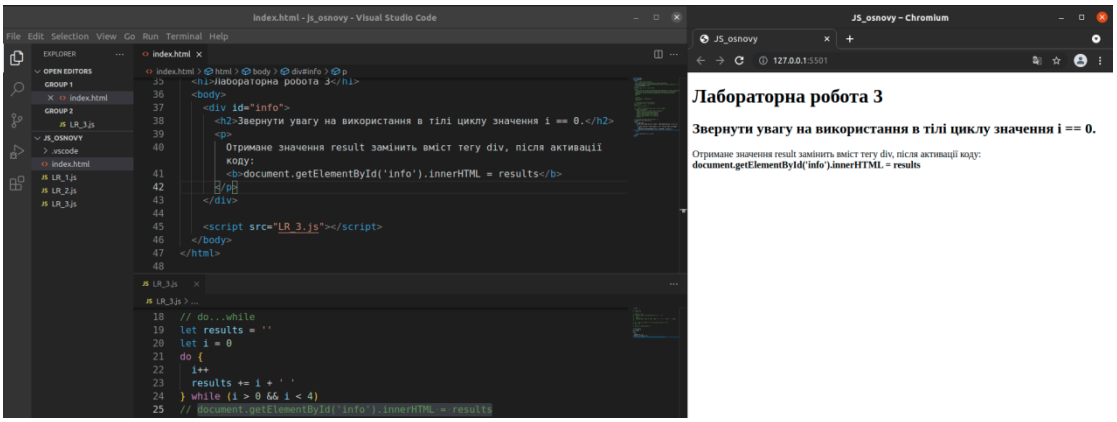

Рисунок 4.3 – Приклад коду з використання do...while без виведення результату

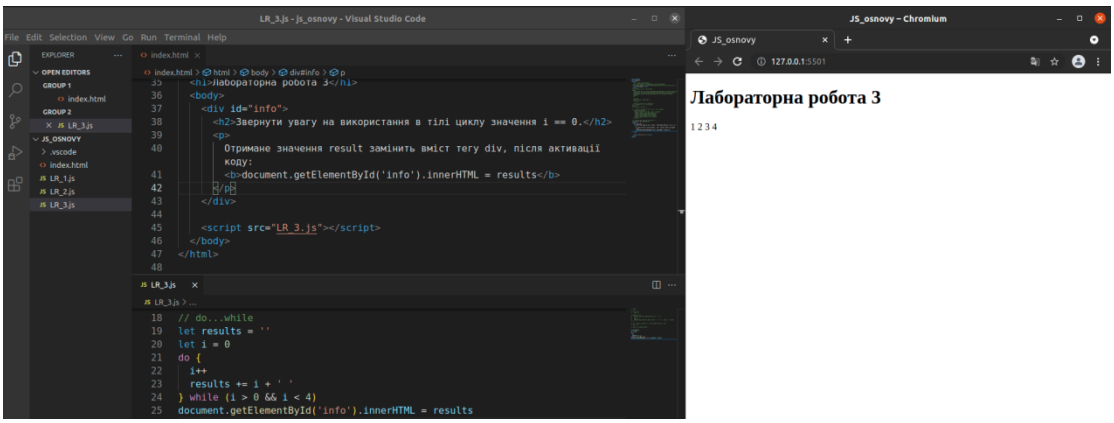

Рисунок 4.4 – Приклад виведення результату виконання do...while

Додатково: document.getElementById () - повертає посилання на елемент за його ідентифікатором (ID) (детальна інфомація за посиланням:
https://developer.mozilla.org/ru/docs/Web/API/Document/getElementById);
властивість innerHTML дозволяє отримати HTML-вміст елемента у вигляді рядка, а також змінювати його (детальна інфомація за посиланням:
https://developer.mozilla.org/ru/docs/Web/API/Element/innerHTML).

4.1.3 Цикл for

В таблиці 4.3 продемонстровані варіанти синтаксису циклу **for**, на рис. 4.5 - 4.9 наведені приклади використання.

Таблиця 4.3 – Варіанти синтаксису циклу for

Варіанти оператора	Синтаксис	Особливості синтаксису
for	`for (початок; умова; крок) { // ... тіло циклу ... }`	ініціалізація – вираз (в тому числі вирази присвоєння) або визначення змінних; використовується, щоб ініціалізувати лічильник; може опціонально оголошувати нові змінні за допомогою ключового слова let; умова – якщо даний вираз істинний, цикл виконується; умова не є обов'язковою (в разі відсутності умова вважається істиною); якщо хибний, виконання переходить до коду за for; фінальний вираз – виконується в кінці ітерації циклу; виконується до наступного виконання умови; використовується для поновлення або збільшення змінної лічильника; вираз – виконується, коли умова циклу істинна; щоб не виконувати ніякого виразу в циклі, використовуйте вираз (;).
Скорочена форма for	`for ([ініціалізація]; [умова]; [фінальний вираз]) вираз`	

Продовження табл. 4.3

Варіанти оператора	Синтаксис	Особливості синтаксису

Будь-яка частина for може бути пропущена.	`let i = 0`
	`for (; i < 2; i++) {`
	`  alert( i )`
	`}`
		початок оголошено вище

	`let i = 0`
	`for (; i < 2;) {`
	`  alert( i++ )`
	`}`
		крок вказаний в тілі циклу

	`for (;;) {` `}`	безкінечний цикл

for...in	`for (variable in object) {`
	`...`
	`}`
		проходить по перерахованим властивостям об'єкта (по кожному окремому елементу);
		variable – (чергове) ім'я властивості призначається змінній на кожній ітерації;
		object – об'єкт по властивостям якого проходимо.
		Зауваження: for ... in не доцільно використовувати для Array, де важливий порядок індексів.

for...of	`for (variable of iterable) {`
	`  statement`
	`}`
		виконує цикл обходу ітерованих об'єктів (включаючи Array, Map, Set, об'єкт аргументів), викликаючи на кожному кроці ітерації оператори для кожного значення з різних властивостей об'єкта;
		на кожному кроці ітерації variable присвоюється значення нової властивості об'єкта iterable; змінна variable може бути оголошеною за допомогою const, let, var;
		iterable – об'єкт по, перерахованим властивостям якого, проходять під час виконання циклу.

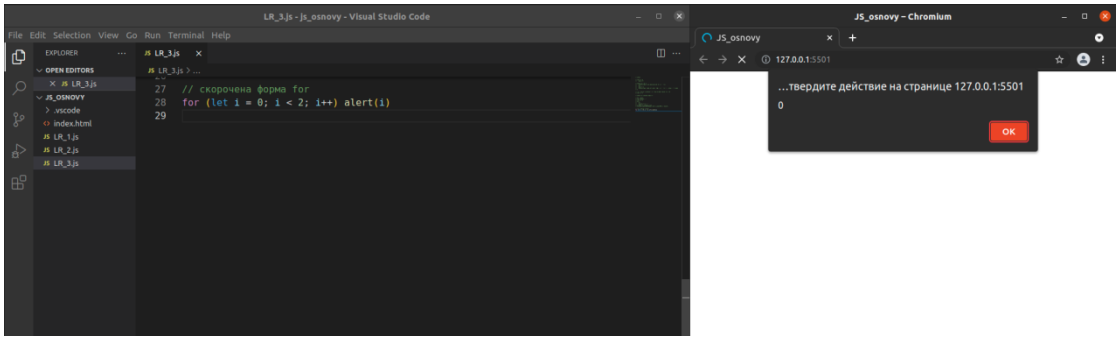

Рисунок 4.5 – Приклад коду з використання скороченої форми for: виведення першого результату

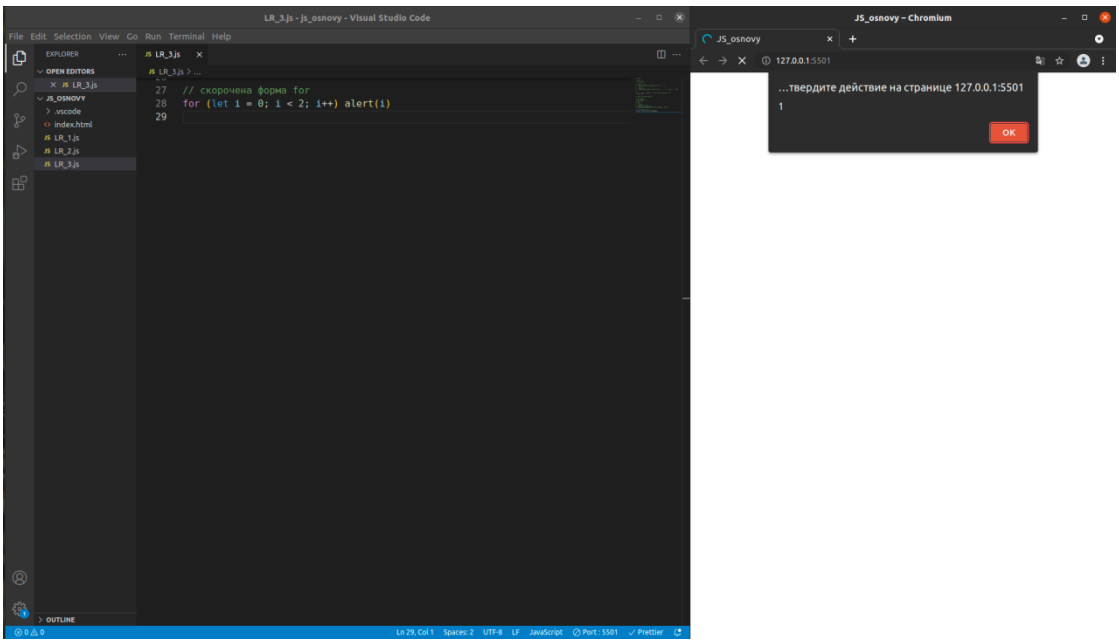

Рисунок 4.6 – Приклад коду з використання скороченої форми for: виведення другого результату

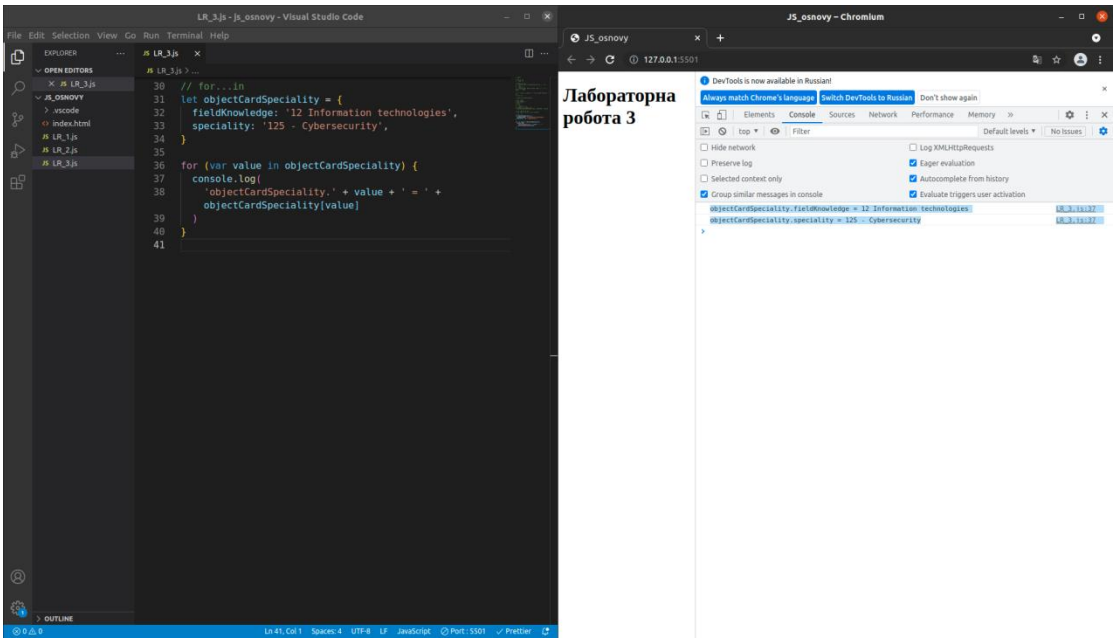

Рисунок 4.7 – Приклад коду з використання for...in

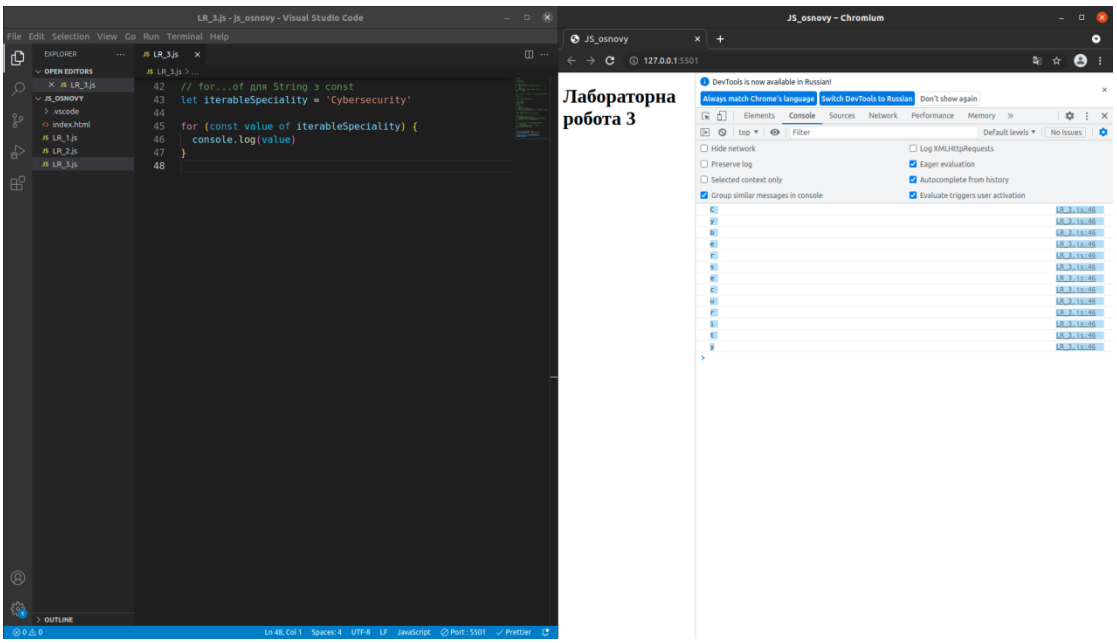

Рисунок 4.8 – Приклад коду з використання for...of для String з const

Можна також використовувати const замість let, якщо не потрібно перепризначувати змінні всередині блоку.

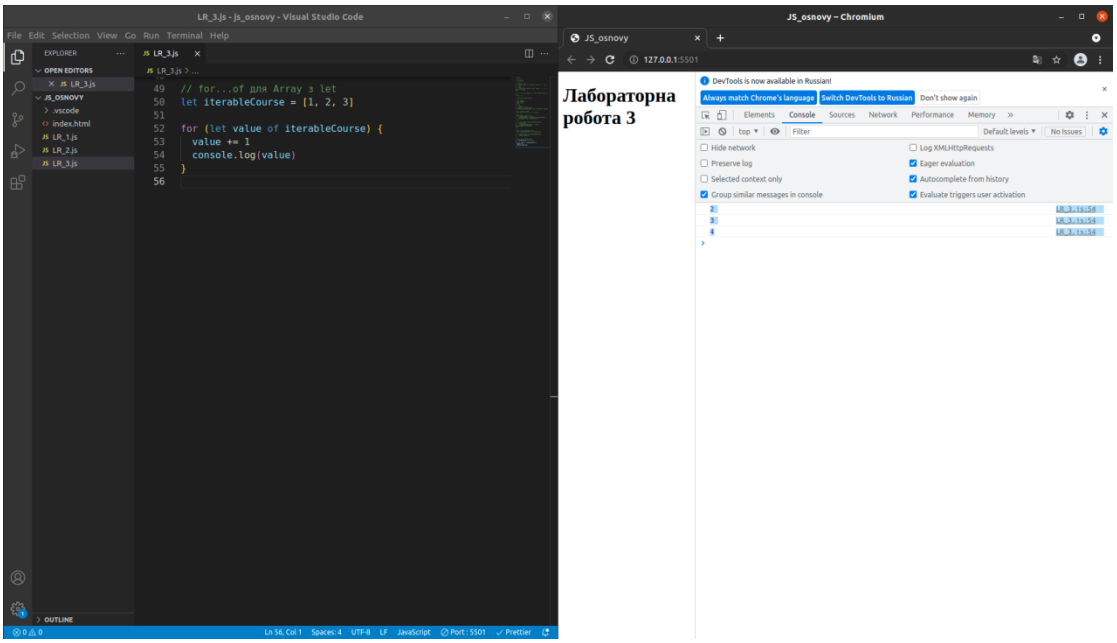

Рисунок 4.9 – Приклад коду з використання for...of для Array з let

4.1.4 Оператор break

Оператор **break** перериває виконання поточного циклу, оператора множинного вибору switch або блочного виразу з міткою. Виконання коду триває з конструкції, наступної після перерваної (рис. 4.10, 4.11, 4.12).

Синитаксис:

```
break [label]
```

де label – необов'язковий ідентифікатор пов'язаної мітки. Якщо переривається вираження не цикл або switch, вказівка мітки обов'язкова.

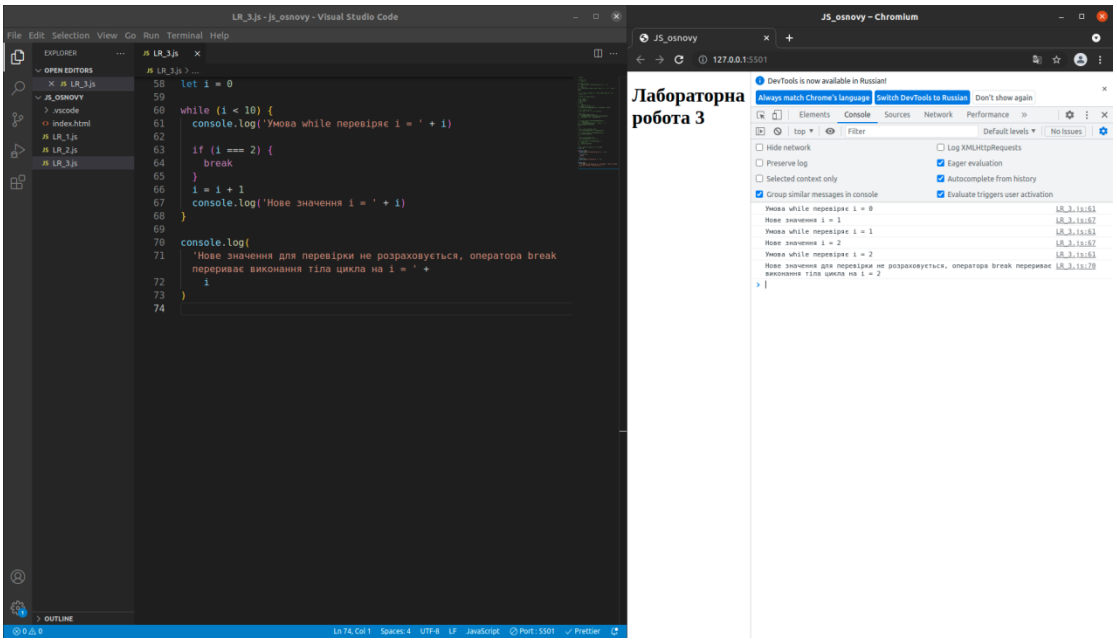

Рисунок 4.10 – Приклад переривання виконання поточного циклу

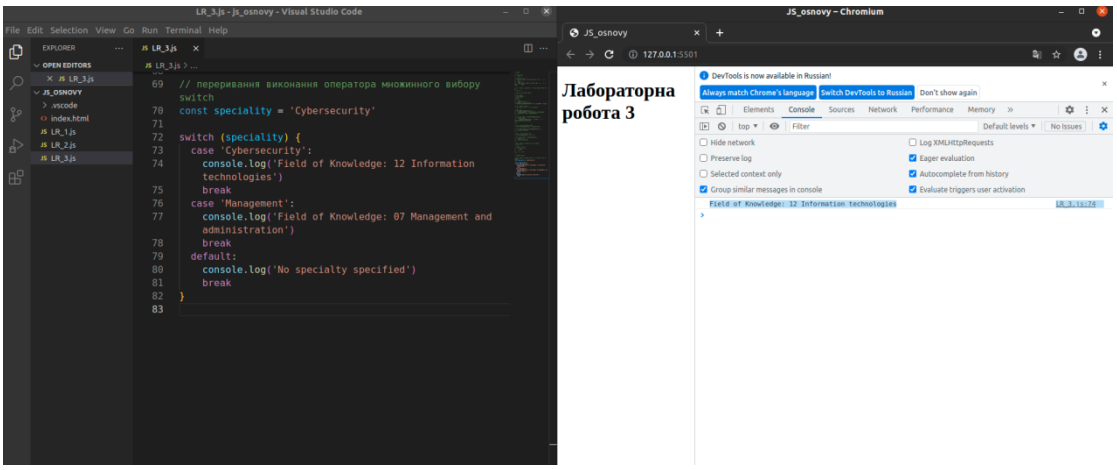

Рисунок 4.11 – Приклад переривання виконання оператора множинного вибору switch

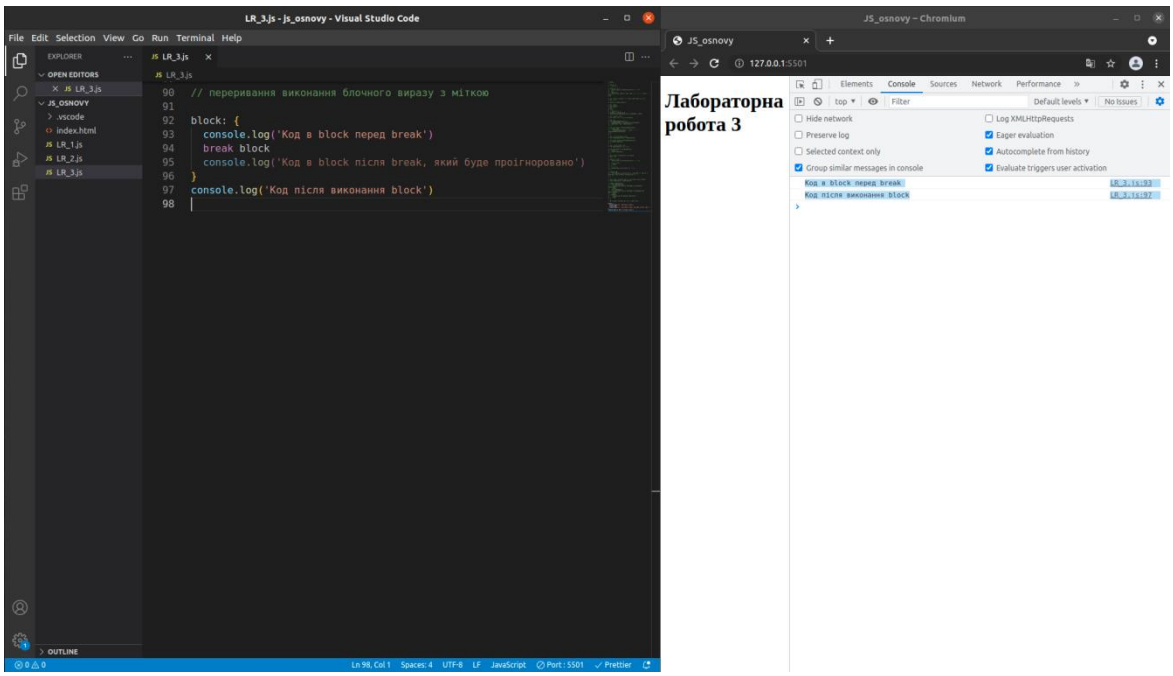

Рисунок 4.12 – Приклад переривання виконання блочного виразу з міткою

4.1.5 Інструкція continue

Інструкція **continue** перериває виконання поточної ітерації поточного каналу або циклу, і продовжує його виконання на наступній ітерації.

Синтаксис:

```
continue [label];
```

де label – ідентифікатор, що є міткою (label) інструкції.

У порівнянні з інструкцією break, continue перериває виконання циклу не повністю, замість цього: в циклі while воно переносить потік виконання до умови; в циклі for воно переносить потік виконання до фінального вислову в описі циклу.

Інструкція continue може використовуватися разом з необов'язковою міткою, яка буде починати наступну ітерацію зазначеного циклу, а не поточного. В даному випадку, continue повинен перебувати всередині зазначеного блоку, який відповідає мітці (рис. 4.13, 4.14).

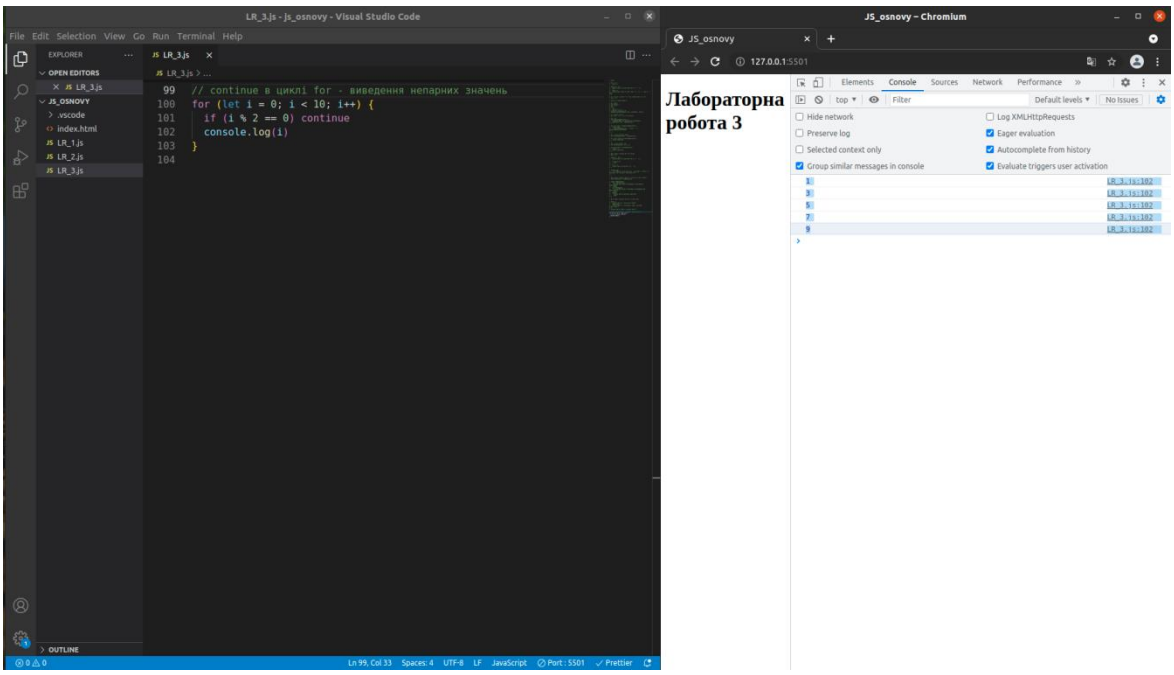

Рисунок 4.13 – Приклад коду з циклом for, який використовує continue для виведення лише непарних значень

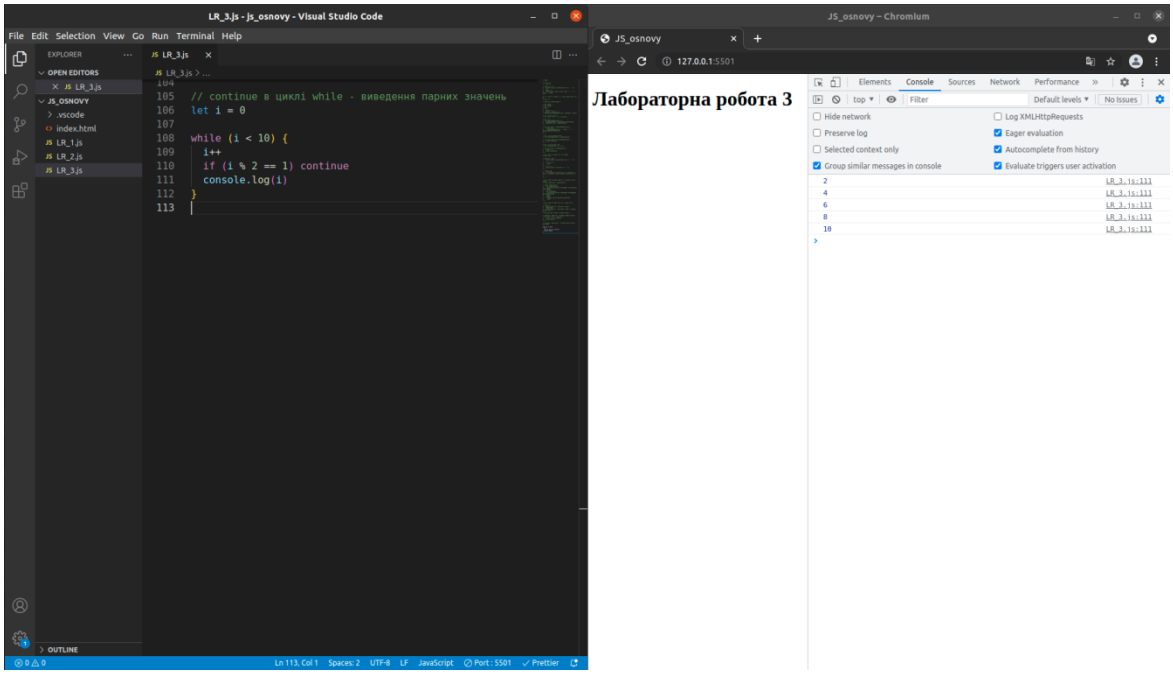

Рисунок 4.14 – Приклад коду з циклом while, який використовує continue для виведення лише парних значень

4.1.6 Мітка

Мітки (label) використовуються разом з операторами break і continue. Вони виступають в ролі ідентифікатора виразу, на який можна ссилатися (рис. 4. 15).

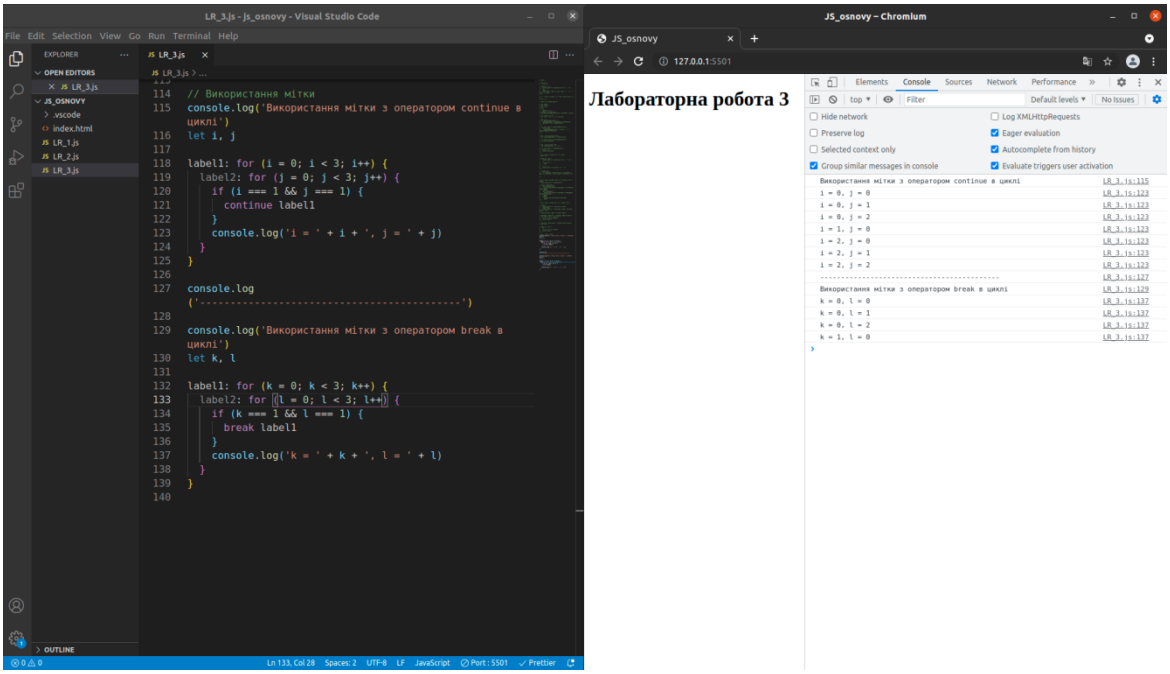

Рисунок 4.15 – Приклад коду з використанням міток

4.2 Завдання до лабораторної роботи

Завдання 1. В проект підключити скрипт лабораторної роботи.

Завдання 2. Написати код виконання задачі, умови яких запропоновано поваріантно (див. Додаток В). В коді:

1) використати цикли;

2) за потреби використати: інструкції break, continue, умовні оператори.

ЛАБОРАТОРНА РОБОТА № 5
ФУНКЦІЇ. АНОНІМНІ ФУНКЦІЇ. ФУНКЦІЇ-КОЛБЕКИ СТРІЛОЧНІ ФУНКЦІЇ. ФУНКЦІЇ-ГЕНЕРАТОРИ

Мета: ознайомитись із способами оголошення функцій, викликом функцій, функціями-колбеками, анонімними, стрілочними функціями, функціями-генераторами.

5.1 Теоретичні відомості

Будь-яка **функція** це об'єкт, тому нею можна маніпулювати як об'єктом, зокрема: передавати як аргумент і повертати як результат під час виклику інших функцій (функцій вищого порядку); створювати анонімно і надавати значень змінних чи властивостей об'єктів.

Приклади вбудованих функцій: alert(message), prompt(message, default) і confirm(question).

5.1.1 Способи оголошення функцій

В таблиці 5.1 продемонстровані варіанти способів оголошення функцій.

Таблиця 5.1 – Варіанти синтаксису способів оголошення функцій

Види	Синтаксис	Особливості синтаксису

конструктор Function,
 	
Function Declaration (оголошення функції)
	`function name ([param,[, param,[..., param]]]) {`
	`  [statements]`
	`}`
	
	`function ім'я (параметри) {`
	`  ...тіло...`
	`}`
	
	`function ім'я (параметри) {`
	`  return інстукція;`
	`}`
		function – ключове слово;
		name – ім'я функції;
		paramN – список параметрів (приймаються функцією) вкладених у круглі дужки () та розділених комами;
		statements – інструкції, становлять тіло функції, вкладені в фігурні дужки { }, будуть виконані після виклику функції;
		інструкція return – вказує на те, яке значення повертати.

Function Expression (функціональний вираз)	`let myFunction = function [name]([param1[, param2[, ..., paramN]]]) { statements};`	name – ім'я функції (може бути упущене, тобто функція є анонімною); paramN – ім'я аргументу, що передається у функцію; statements – інструкції, тіло функції.

Якщо функцію оголошено як окрему інструкцію в основному потоці коду, це **Function Declaration** (рис. 5.1, 5.3). Function Declaration обробляються перед виконанням кодового блоку; їх видно у всьому блоці; виклик функції піднімає визначення функції (**hoisting**) (рис. 5.2).

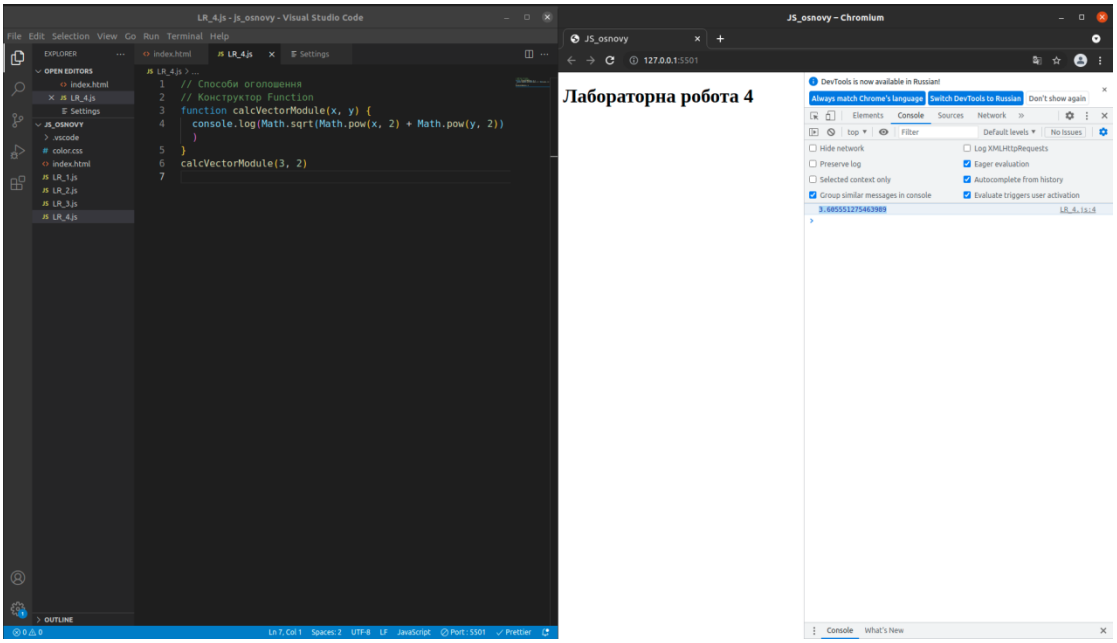

Рисунок 5.1 – Приклад використання конструктора Function

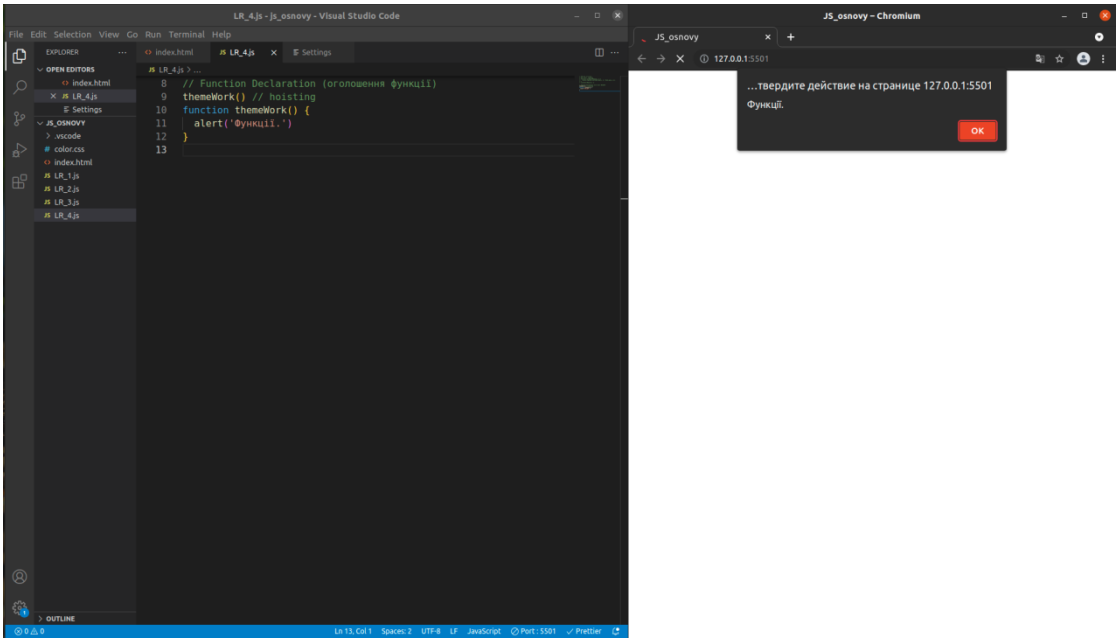

Рисунок 5.2 – Приклад використання Function Declaration (hoisting)

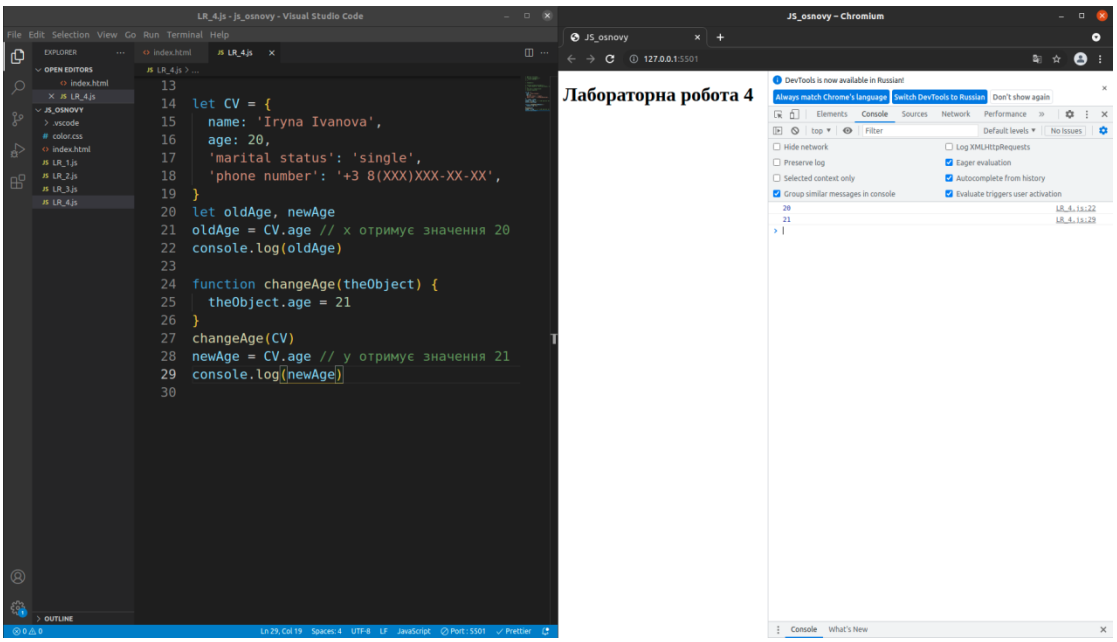

Рисунок 5.3 – Приклад функції, яка змінить властивість переданого до неї об'єкта

За замовчуванням функції повертають undefined. Щоб повернути інше значення, функція може містити інструкцію **return**, яка вказує на те, яке значення повертати (рис. 5.4).

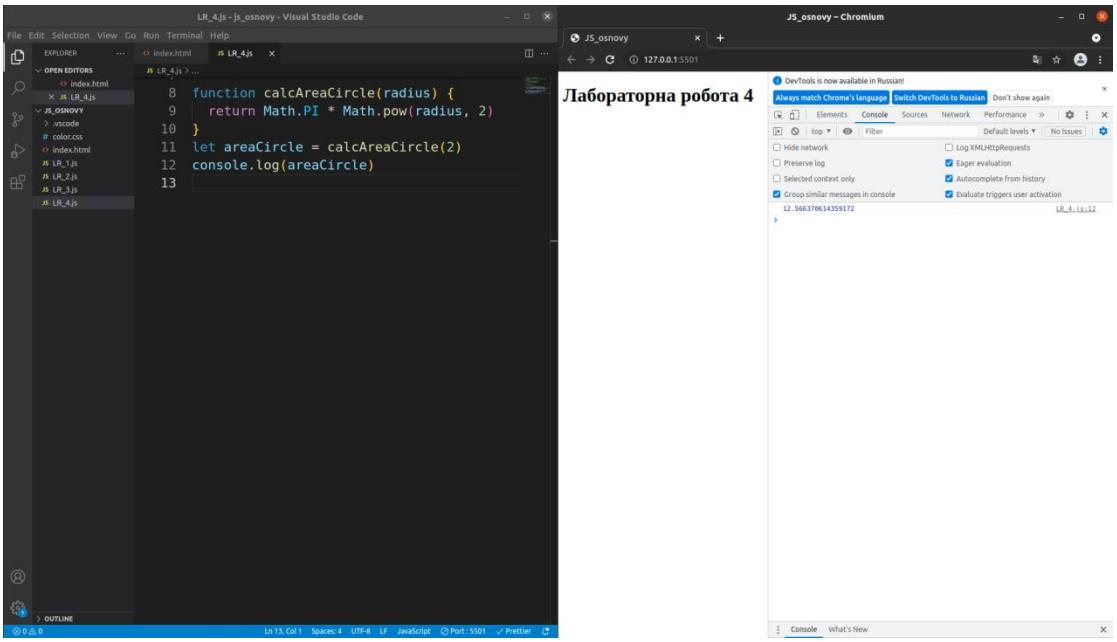

Рисунок 5.4 – Приклад використання інструкції return

Якщо функцію було створено як частину виразу, то вважається, що функцію оголошено за допомогою **Function Expression** (рис. 5.5 - 5.7). Функції, оголошені за допомогою Function Expression, створюються лише коли потік виконання досягає їх; визначення функції не піднімаються (**not hoisting**) (рис. 5.6).

Ключова ідея **функцій-«колбеків»** в тому, що передається функція і очікуєтья, що вона викличеться назад (від англ. call back - зворотний виклик) пізніше, якщо це буде необхідно (рис. 5.8, 5.9).

Анонімні функції використовуються як колбек-функції; ім'я може бути і присвоєно для виклику самої себе всередині самої функції та для відладчика (debugger) для ідентифікованих функцій у стек-треках (stack traces; "trace" - "слід" / "відбиток") (рис. 5.5).

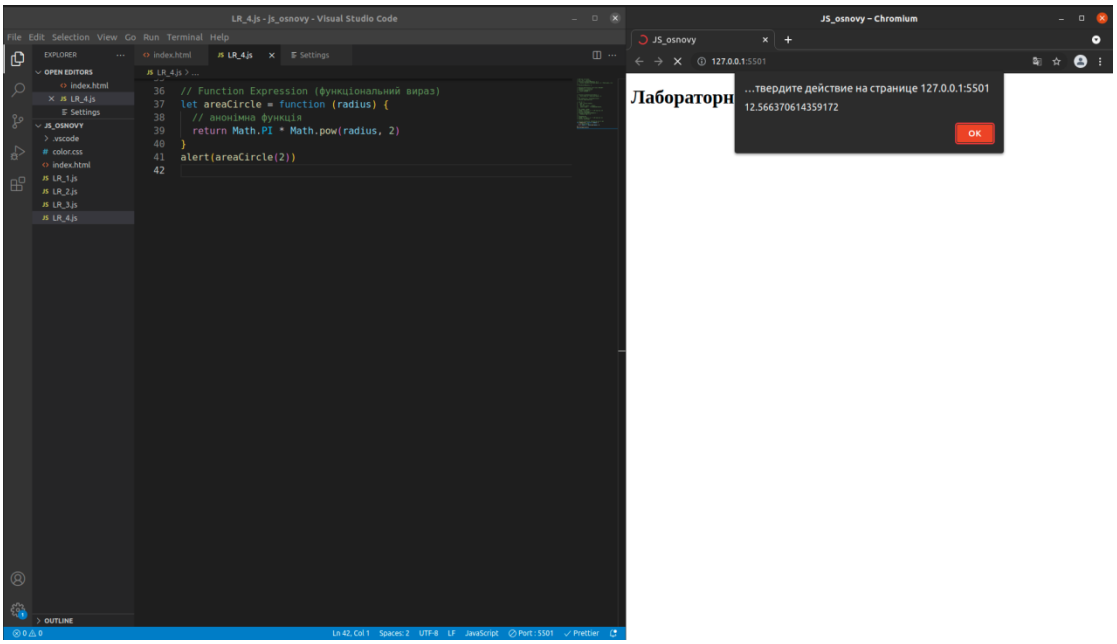

Рисунок 5.5 – Приклад використання анонімної функції через Function Expression

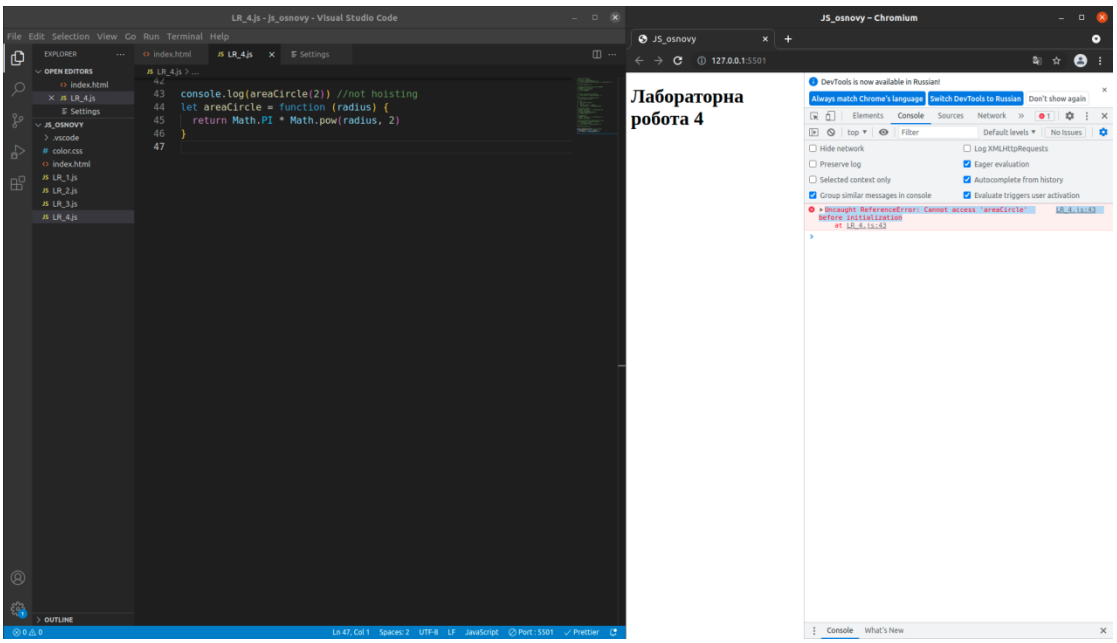

Рисунок 5.6 – Повідомлення про помилку під час виклику функції до її оголошення через Function Expression

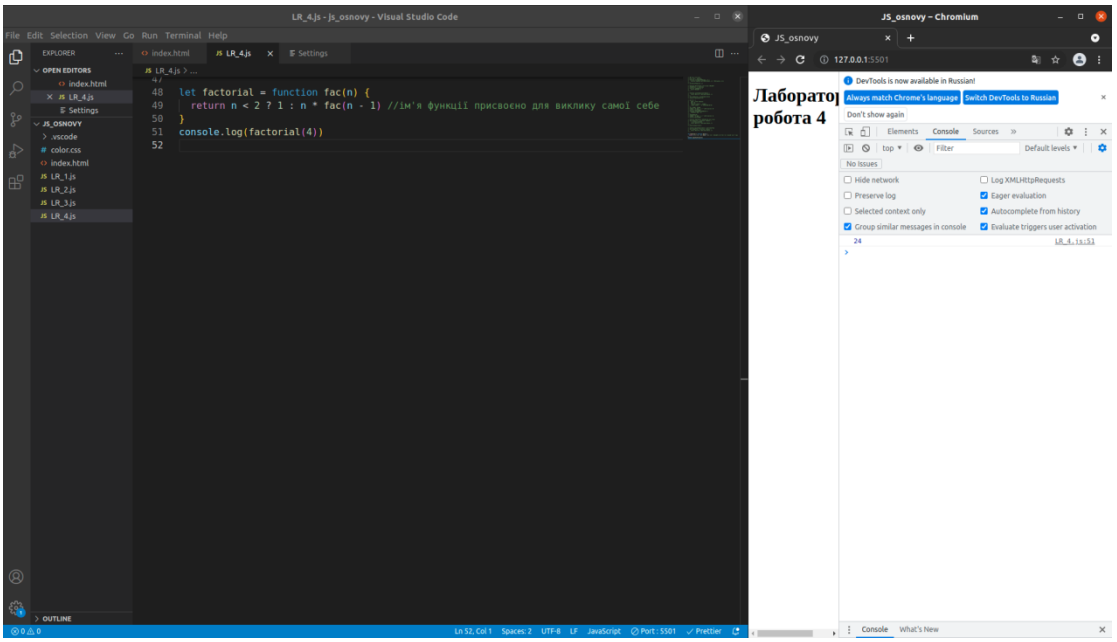

Рисунок 5.7 – Приклад Function Expression, де ім'я функції присвоєно для виклику самої себе

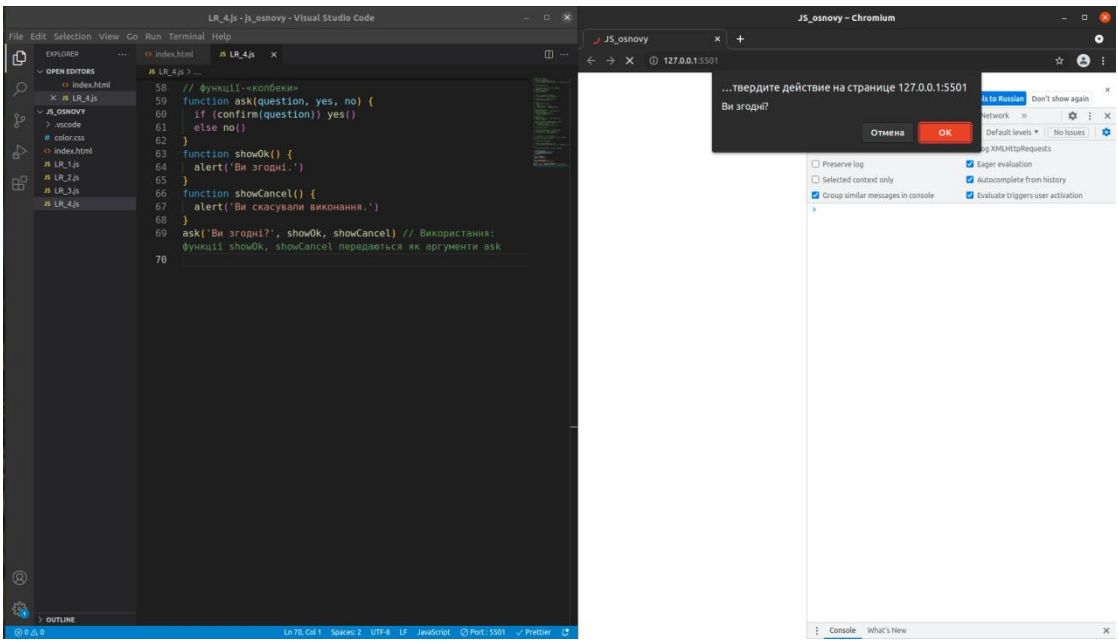

Рисунок 5.8 – Приклад функції-«колбеку»: запит для користувача

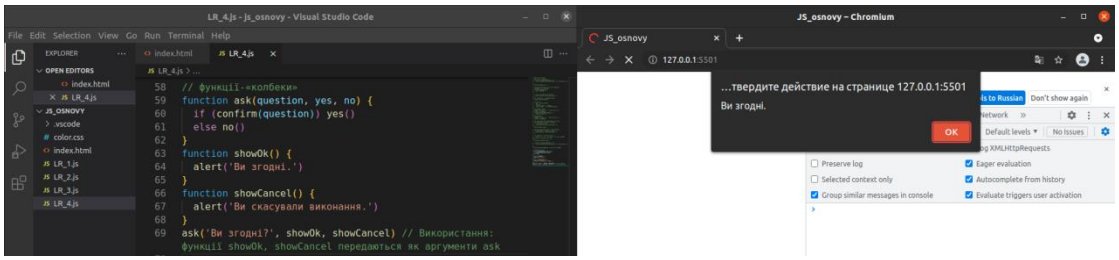

Рисунок 5.9 – Приклад функції-«колбеку»: showOk колбек для відповіді «yes»

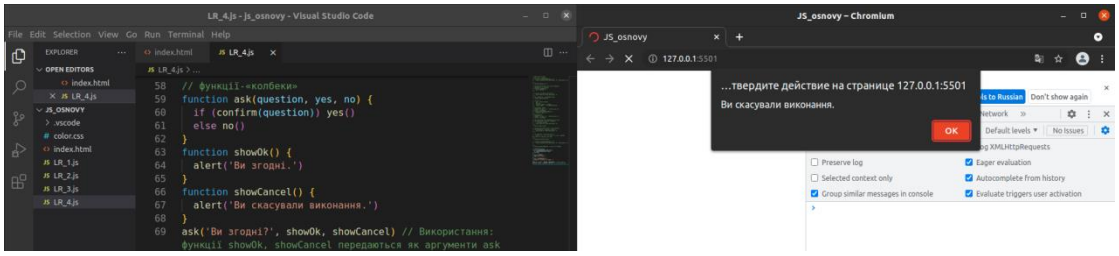

Рисунок 5.10 – Приклад функції-«колбеку»: showCancel колбек для відповіді «no»

Насправді все одно, як ми визначили функцію, це просто значення, що зберігається в змінній. Сенс обох способів однаковий: "створити

функцію та помістити її значення в змінну", вивести дане значення можна за допомогою alert (рис. 5.11).

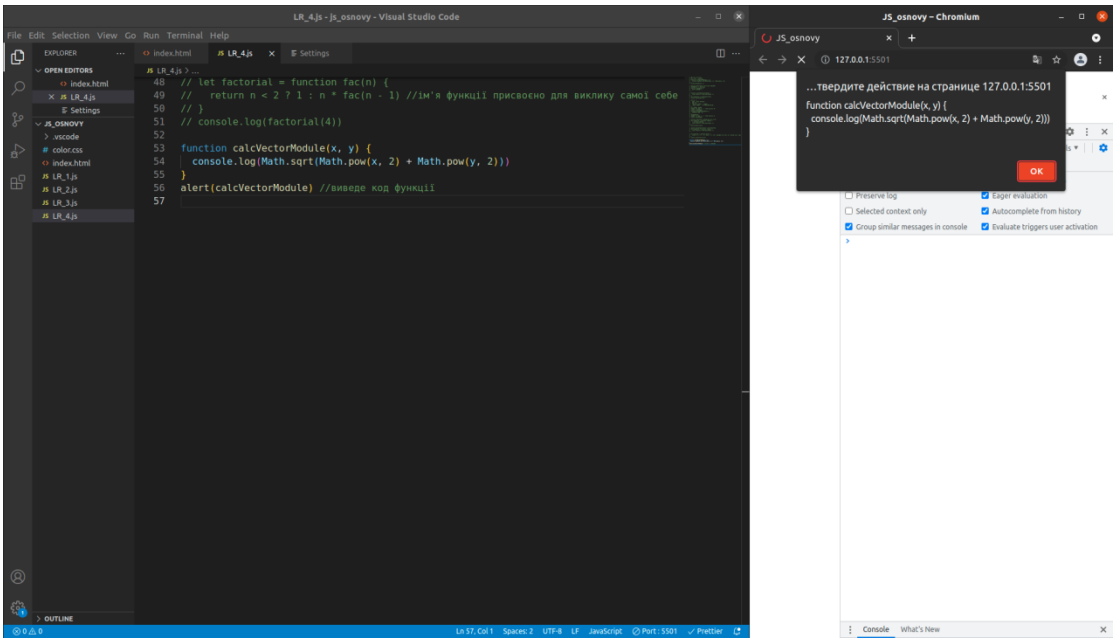

Рисунок 5.11 – Приклад виведення коду функції

5.1.2 Способи виклику функцій

Варто зауважити, що оголошення функції не виконує її. Оголошення функції називає функцію і вказує, що робити під час виклику функції.

Виклик функції фактично виконує вказані дії із зазначеними параметрами. Наприклад, як подано в одному з прикладів (рис. 5.1):

```
calcVectorModule(3, 2)
```

Існують інші способи викликати функцію, наприклад, поширені випадки, коли функції необхідно викликати динамічно, або змінити номери аргументів функції, або викликати функцію з прив'язкою до певного контексту. Виявляється, що функції самі по собі є об'єктами, і ці об'єкти, у свою чергу, мають методи. Один із них це метод apply(), використання якого може досягти цієї мети.

Вибір імені функції

Ім'я функції зазвичай є дієсловом. Як правило, використовуються дієслівні префікси, що позначають загальний характер дії, після яких слідує уточнення.

Зазвичай у командах розробників діють угоди щодо значень цих префіксів. Функції, що починаються з, наприклад: "show" – зазвичай виконують демонстрацію;

"get…" – повертають значення; "calc…" – винують обчислення; "create…" – створюють; "check…" – виконують перевірку та повертають логічне значення, тощо.

Бажано, щоб функція виконувала лише те, що очевидно передбачає її назва, причому однією дією. Дві незалежні дії зазвичай передбачають дві функції, навіть якщо вони будуть викликатися разом (у цьому випадку можна створити третю функцію, яка їх викликатиме).

5.1.3 Параметри та аргументи виклику (arguments)

У JavaScript параметри за замовчуванням мають значення undefined. В таблиці 5.2 продемонстровані деякі види параметрів, на рис. 5.12, 5.13 відповідні приклади.

Таблиця 5.2 – Види параметрів

Види	Синтаксис	Призначення
Параметри за замовчуванням (Default parameters)	`function [name] ([param1[ = defaultValue1 ][, ..., paramN[ = defaultValueN ]]]) { Statements }`	дозволяють задавати формальні параметри функції, тобто значення за замовчуванням у випадку, якщо функція викликана без аргументів, або якщо параметр явно передано значення undefined
Залишкові параметри (Rest parameters)	`function(a, b, ...theArgs) { Statements }`	дозволяє представляти безліч аргументів у вигляді масиву

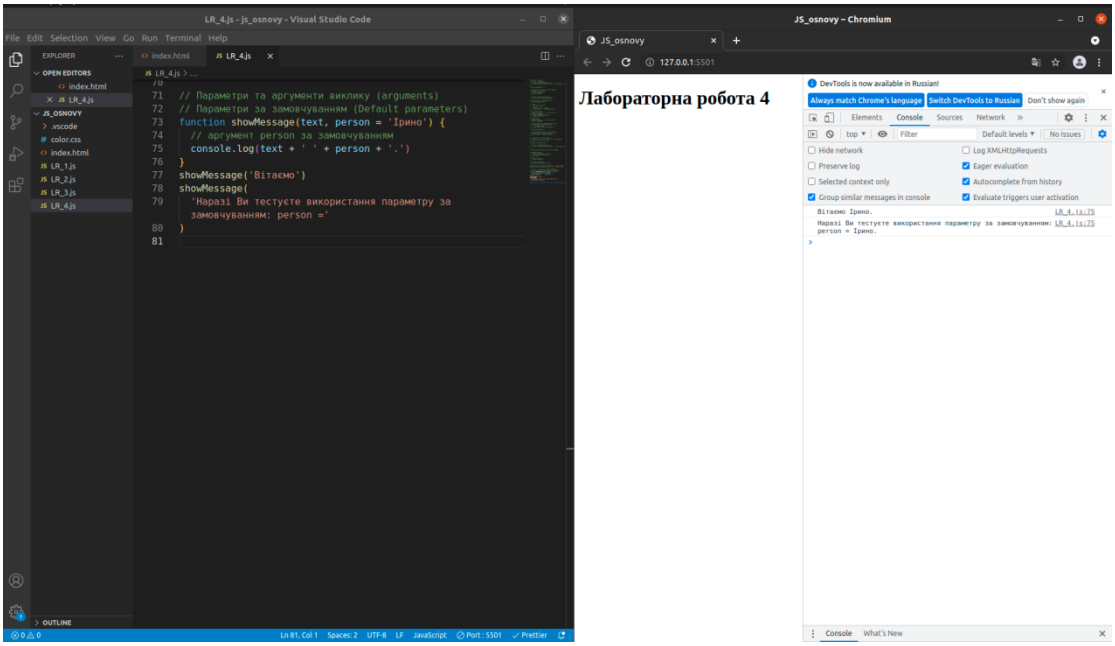

Рисунок 5.12 – Приклад використання параметрів за замовчуванням

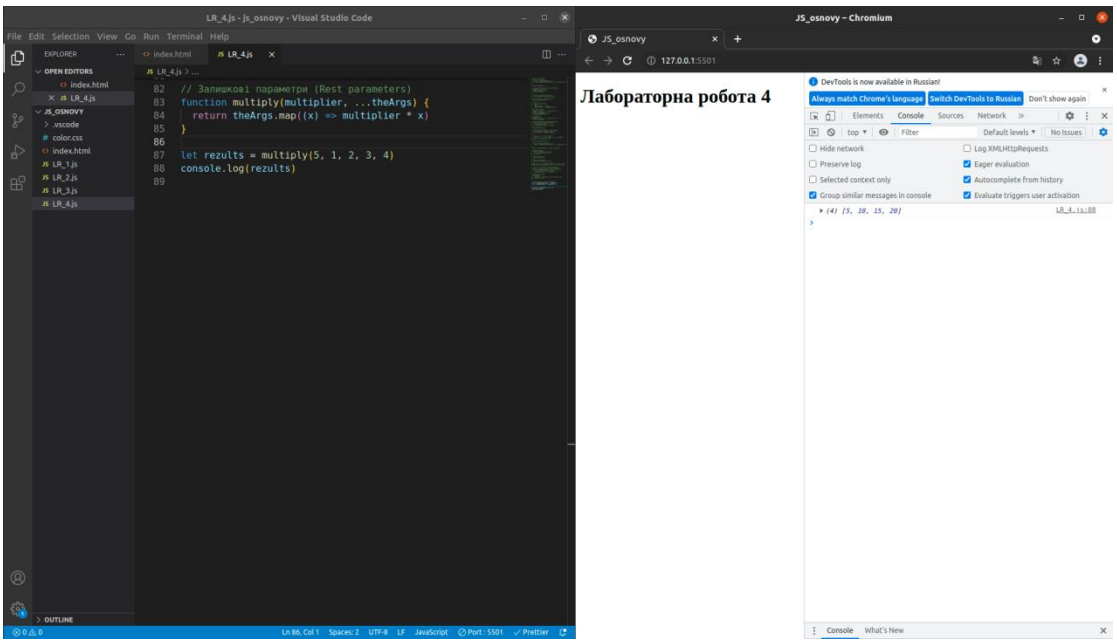

Рисунок 5.13 – Приклад використання залишкових параметрів

5.1.4 Область видимості функцій (function scope)

Змінні оголошені в функції не можуть бути доступними поза межами даної функції, тому змінні (які потрібні саме для функції) оголошують тільки в scope функції – **локальні змінні.** При цьому функція має доступ до всіх змінних та функцій, оголошених усередині її scope. Тобто функція оголошена у глобальному scope має доступ до всіх змінних у глобальному scope. Функція оголошена всередині іншої функції ще має доступ і до всіх змінних батьківської функції та інших змінних, до яких ця батьківська функція має доступ.

Глобальні змінні – змінні, оголошені зовні всіх функцій. Глобальні змінні видимі для будь-якої функції (якщо їх не перекривають однойменні локальні змінні). Бажано зводити використання глобальних змінних до мінімуму. У сучасному коді зазвичай мало чи зовсім немає глобальних змінних. Хоча вони іноді корисні для зберігання найважливіших «загальнопроектових» даних.

5.1.5 Стрілочні функції

Вирази **стрілочних функцій** мають короткий синтаксис порівняно з функціональними виразами. Вираз стрілочних функцій не дозволяє задавати ім'я, тому зазвичай стрілочні функції анонімні.

Вираз стрілочних функцій не може містити розриви рядків між параметрами та стрілкою.

В таблиці 5.3 продемонстровані види синтаксису стрілочних функцій.

Таблиця 5.3 – Синтаксис стрілочних функцій

Синтаксис	Коментар
`(param1, param2, …, paramN) => { statements }`	Базовий синтаксис

`(param1, param2, …, paramN) => expression`
// еквівалентно:
`(param1, param2, …, paramN) => { return expression; }`
	// Короткий синтаксис, що неявно повертає результат
	// Блоковий синтаксис, явно повертає результат

`singleParam => { statements }`
// еквівалентно:
`(singleParam) => { statements }`
	Круглі дужки не є обов'язковими для єдиного параметра

`() => { statements }`
`() => expression`
// Еквівалентно:
`() => { return expression;}`
	// Функція без параметрів потребує круглих дужок

`params => ({foo: bar})`	Розширений синтаксис // Коли повертаєте літеральний вираз об'єкта, покладіть тіло в дужки

`(param1, param2, ...rest) => { statements }`
 	
`(param1,…, paramN = defaultValueN) => { statements }`
	// Залишкові параметри та параметри за замовчуванням підтримуються

`let f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c` `f() // 6`	// Деструктуризація також підтримується

5.1.6 Функції-генератори

function* (ключове слово function із зірочкою) визначає функцію-генератор.

Ключове слово function* може бути використане для оголошення функції-генератора всередині виразу.

Вираз function* дуже схожий на оголошення function*, і має майже однаковий синтаксис. Основна різниця між виразом function* та

оголошенням function* – ім'я функції, яке може бути відсутнє у виразах function* для створення анонімних функцій (табл. 5.4).

Таблиця 5.4 – Синтаксис функції-генератора

Види	Синтаксис	Особливості синтаксису
оголошення function*	`function* name([param[, param[, ... param]]]) { statements }`	name – ім'я функції; param – іменовані аргументи функції (параметри); функція-генератор може мати 255 аргументів; statements – інструкції, складові функції тіла
вираз function*	`function* [ім'я]([параметр1[, параметр2[, ..., параметрN]]]) { інструкції }`	ім'я функції, може бути відсутнє, у такому разі функція буде анонімною

4.1.7 Метод

Метод - це функція, асоційована з об'єктом або, метод - це властивість об'єкта, що є функцією. Методи визначаються так само, як і звичайні функції, за винятком, що вони присвоюються властивості об'єкта. Наприклад:

```
objectName.methodname = function_name;
let myObj = {
  myMethod: function(params) {
     // ...do something
  }
}
```

де objectName – це існуючий об'єкт, methodname – це ім'я методу, і function_name – це ім'я безпосередньо функції.

Потім можна викликати метод у контексті об'єкта таким чином:

```
object.methodname(params)
```

Метод **apply()** викликає функцію із зазначеним значенням this та аргументами, наданими у вигляді масиву (або масивоподібного об'єкта).

Примітка: хоча синтаксис цієї функції практично повністю ідентичний функції call(), фундаментальна різниця між ними полягає в тому, що функція call() приймає список аргументів, тоді як функція apply() приймає одиничний масив аргументів.

Синтаксис:

```
fun.apply(thisArg, [argsArray])
```

де thisArg – опціональний параметр; значення this, яке надається для виклику функції fun;
argsArray – опціональний параметр; масивоподібний об'єкт, що визначає аргументи, з якими функція fun повинна бути викликана, або null або undefined, якщо в функцію не треба передавати аргументи.

Метод **eval()** виконує JavaScript-код, поданий рядком.

Важливо: eval() – небезпечна функція, яка виконує код, що проходить з усіма привілеями того, хто її викликає. Якщо запускається eval() з рядком, на який можуть впливати зловмисники, можна запустити шкідливий код на пристрій користувача з правами веб-сторінки/розширення.

Синтаксис:

```
eval (string)
```

де string – рядок представлений JavaScript виразом, оператором або послідовністю операторів. Вираз може містити змінні та властивості існуючих об'єктів.

Приклади:

```
console.log(eval('2 + 2')) // expected output: 4
console.log(eval(new String('2 + 2'))) // expected output: 2 + 2
console.log(eval('2 + 2') === eval('4')) // expected output: true
```

5.2 Завдання до лабораторної роботи

Завдання 1. В проект підключити скрипт лабораторної роботи.

Завдання 2. Написати код виконання задачі, умови яких запропоновано поваріантно (див. Додаток Д). В коді:

1) використати способи оголошення функції: Function Declaration, Function Expression;

2) протестувати завдання у одному з виглядів: стрілочної функції (arrow function), або функції-«колбеку» (callback function), або функції-генератора (generator function), тощо;

3) за доцільності використати: параметри за замовчуванням (Default parameters), або залишкові параметри (Rest parameters).

Перелік рекомендованої літератури

Базова

1. Boris Cherny. Programming TypeScript: Making Your JavaScript Applications Scale. — O'Reilly Media, 2019. — 324 p.

2. David Flanagan. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. — 7th Edition — O'Reilly Media, 2020. —706 p.

3. HTML, JavaScript, PHP и MySQL. Джентельменский набор Web-мастера [Текст] : научное издание / Николай Прохоренок. - 3-е изд. - СанктПетербург : "БХВ-Петербург", 2017. - 912 с.

4. Marijn Haverbeke. Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming — No Starch Press, 2018. — 472 p. Also available online https://eloquentjavascript.net/

5. Гудман, Д. JavaScript и DHTML. Сборник рецептов. Для профессионалов / Д. Гудман. - М.: Питер, 2015. - 523 с.

6. Дмитриева М.В. JavaScript. Экспрес курс / Дмитриева М.В. — СПб. : БХВ-Петербург, 2004. — 620 с.

7. Дунаев В. Самоучитель Java Script/ В. Дунаев . - 2-е изд.. - СПб.: Питер, 2006. - 395 с.

8. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. – [3-е изд.]. – СПб.: Питер, 2015. – 688 с.

9. Прохоренок, Н. А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Webмастера / Н.А. Прохоренок, В.А. Дронов. - Москва: СПб. [и др.] : Питер, 2015. - 768 с.

10. Роббинс Д.Н. HTML5, CSS3 и JavaScript. Исчерпывающее руководство / Д.Н. Роббинс. – [4-е изд.]. – М.: Эксмо, 2014. – 516 с.

11. Создание Web-страниц: HTM, CSS, YavaScript [Текст] / Игорь Владимирович Мархвида. - Минск : Новое знание, 2016. - 352 с.

12. Флэнаган Д. JavaScript. Подробное руководство / Д. Флэнаган. – [6-е изд.]. – СПб.: Символ-Плюс, 2012. – 1080 с.

Додаткова

1. Jon Duckett. Web Design with HTML, CSS, JavaScript and jQuery Set. — Wiley, 2014. — 1152 p.

2. Будилов В.А. JavaScript, XML и объектная модель документа / Будилов В.А. — СПб. : Наука и техника, 2001. — 291 с.

3. Вайк А. JavaScript в примерах / Вайк А. — К. : ДиаСофт, 2000. — 376 с.

4. Рейсиг Дж. JavaScript. Профессиональные приемы программирования / Дж. Рейсиг. — СПб. : Питер, 2008. — 352 с.

5. Уэнц К. JavaScript. Карманный справочник / К. Уэнц. – Санкт-Петербург: Вильямс, 2007.– 272 с.

Інформаційні ресурси

1. The Modern JavaScript Tutorial [Електронний ресурс] – Режим доступу: https://javascript.info/

2. Введение в JavaScript. – Режим доступу до сайту: http://www.intuit.ru/department/internet/js/

3. Кан. М. Основы программирования на JavaScript. [Електрон. ресурс]. - Режим доступу: http://www.intuit.ru/department/internet/jsbasics/13/ MDN web docs. https://developer.mozilla.org

4. Курс Create a back-end app with JavaScript [Електронний ресурс]. – Режим доступу: https://www.codecademy.com/learn/paths/create-a-back-end-app-withjavascript

5. Ответы на вопросы по JavaScript. [Електрон. ресурс]. - Режим доступу: http://thebesthost.ru/articles/HTML/js1.html

6. Програмування WEB [Електронний ресурс]. – Режим доступу: https://metanit.com/web/

7. Ресурси для розробників, від розробників. [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/

8. Учебник Java-script. [Електрон. ресурс]. - Режим доступу: http://javascript.ru/tutorial Учебник по Java Script. [Електрон. ресурс]. - Режим доступу: http://webdesign.net-soft.ru/js.htm

9. Центральный Javascript-ресурс. Учебник с примерами скриптов. Форум. Книги и многое другое. [Електрониий ресурс]. — Режим доступу : http://learn.javascript.ru/.

ДОДАТКИ

Додаток А

Завдання до виконання лабораторної роботи № 2

Рисунок А.1– Перелік варіантів для першого виразу

№ варіанту	Завдання
1	$\frac{(a+b)^2-(a^2+2ab)}{b^2}$
2	$\frac{(a-b)^2-(a^2-2ab)}{b^2}$
3	$\frac{(a+b)^3-(a^3+3a^2b)}{3ab^2+b^3}$
4	$\frac{(a+b)^3-(a^3)}{3ab^2+b^3+3a^2b}$
5	$\frac{(a-b)^3-(a^3-3a^2b)}{b^3-3ab^2}$
6	$\frac{(a-b)^3-(a^3-3ab^2)}{b^3-3a^2b}$
7	$\frac{(a-b)^3-(a^3)}{b^3-3ab^2-3a^2b}$
8	$\frac{(a+b)^4-(a^4+4a^3b+6a^2b^2)}{4ab^3+b^4}$
9	$\frac{(a+b)^4-(a^4+4a^3b)}{6a^2b^2+4ab^3+b^4}$
10	$\frac{(a-b)^4-(a^4-4a^3b+6a^2b^2)}{b^4-4ab^3}$

Рисунок А.2– Перелік варіантів для другого виразу

1) $y = \cos x \operatorname{tg} x$;	4) $f(x) = x^2 \cos x$;	7) $f(x) = \sin 2x \operatorname{tg} x$;
2) $y = \sin^2 x$;	5) $f(x) = x \sin x^2$;	8) $f(x) = x \sin x$;
3) $y = \sin x \operatorname{tg} x$;	6) $y = \sin x + \operatorname{tg} x$;	9) $f(x) = 1 + \cos x$;
		10) $f(x) = \frac{\operatorname{tg} x}{x}$.

Додаток Б

Завдання до виконання лабораторної роботи № 3

1. Написати програму, яка обчислює частку двох чисел. Програма повинна перевіряти правильність введених користувачем даних і, якщо вони невірні (дільник дорівнює нулю), видавати повідомлення про помилку. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Обчислення частки.
Введіть ділене і дільник, ->12 0
Ви помилилися. Дільник не повинен дорівнювати нулю.

2. Написати програму обчислення опору електричного струму, що складається з двох опорів. Опори можуть бути з'єднані послідовно або паралельно. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Обчислення опору електричного кола.
Введіть вихідні дані:
Величина першого опору (Ом) ->15
Величина другого опору (Ом) ->27.3
*Тип з'єднання (*1 - *послідовне,* 2 - *паралельне)* ->2
Опір ланцюга: 9.68 *Ом*

3. Написати програму, яка переводить час з хвилин і секунд в секунди. Програма повинна перевіряти правильність введених користувачем даних і в разі, якщо дані невірні,виводити відповідне повідомлення. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Введіть час (хвилини.секунди) ->2.90
Помилка! Кількість секунд не може бути більше 60

4. Написати програму, яка перевіряє, чи є рік високосним. Нижче наведено рекомендований вигляд екрану під час роботи програми.

Введіть рік, наприклад 2000, ->2001
2001 рік - не високосний

5. Написати програму розв’язання квадратного рівняння. програма повинна перевіряти правильність вихідних даних і в випадку, якщо коефіцієнт при другому ступені невідомого дорівнює нулю, виводити відповідне повідомлення. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Розв’язання квадратного рівняння
Введіть значення коефіцієнтів ->12 27 -10
Корені рівняння:
x_1 = -25.551
x_2 = -28.449

6. Написати програму обчислення вартості покупки з урахуванням знижки. Знижка в 10% надається, якщо сума покупки більше 1000 грн.

Нижче наведено рекомендований вигляд екрану підчас виконання програми.

Обчислення вартості покупки з урахуванням знижки
Введіть суму покупки (грн.) ->1200
Вам надається знижка 10%
Сума покупки з урахуванням знижки: 1080.00*грн.*

7. Написати програму, яка обчислює оптимальну вагу для користувача, порівнює його з реальним і видає рекомендацію про необхідність набрати вагу або схуднути. Оптимальна вага обчислюється за формулою: Зріст (см) - 100. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Введіть зріст (см) і вагу (кг), ->170 68
*Вам треба набрати вагу*2.00 *кг.*

8. Напишіть програму, яка запитує у користувача номер місяця і потім виводить відповідну назву часу року. У разі, якщо користувач введе неприпустиме число, програма повинна вивести повідомлення "Помилка введення даних ". Нижче наведено рекомендований вигляд екрану під час роботи програми.

Введіть номер місяця
->11
зима

9. Написати програму, яка запитує у користувача номер дня тижня і виводить одне з повідомлень: "Робочий день", "Субота" або "Неділя".

10. Напишіть програму перевірки знання історії архітектури. Програма повинна вивести питання і три варіанти відповіді. Користувач повинен вибрати правильну відповідь і ввести його номер. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Архітектор Ісаакіївського собору:
1. Доменіко Трезини
2. Огюст Монферран
3. Карл Россі
Введіть номер правильної ->3
Ви помилилися.
Архітектор Ісаакіївського собору - Огюст Монферран.

Додаток В

Завдання до виконання лабораторної роботи № 4

1. Написати програму, яка обчислює суму перших *n*членів ряду: 1, 3, 5, 7 ... Кількість членів ряду, які будуть шумуватись,задається під час роботи програми. Нижче наведено рекомендований вигляд екрану під час роботи програми.

Обчислення часткової суми ряду: 1,3,5,7 ...
Введіть кількість членів ряду для сумування ->15
Сума перших 15 *членів ряду дорівнює* 330

2. Написати програму, яка виводить таблицю значень функції $y = |x-2| + |x+1|$. Діапазон зміни аргументу від -4 до 4,крокзбільшення аргументу 0,5.

3. Напишіть програму, яка виводить на екран квадрат Піфагора – таблицю множення. Рекомендований вигляд екрану:

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90

4. Написати програму, яка перетворює введене користувачем десяткове число в двійкове. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Перетворення десяткового числа в двійкове
*Введіть ціле число від*0*до* 255 ->49
Десятковому числу 49*відповідаєдвійкове*00110001

5. Написати програму, яка обчислює факторіал введеного з клавіатури числа. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Обчислення факторіала.
Введіть число, факторіал якого треба вирахувати->7
Факторіал 7 *дорівнює* 5040

6. Напишіть програму, яка перевіряє, чи є введене користувачем ціле число простим. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Введіть ціле число і натисніть ->45
45 - *не просте число.*

7. Написати програму наближеного обчислення інтеграла методом трапецій. Після кожного циклу обчислень програма повинна виводити обчислене значення, кількість і величину інтервалів.

8. Напишіть програму, яка виводить на екран таблицю значень функції $y = -2{,}1x^2 - 0{,}24x + 1{,}6$ в діапазоні від -4 до 4. Крок зміни аргументу 0,5.

9. Напишіть програму, яка обчислює число " π " із заданою користувачем точністю. Для обчислення значення числа "Пі" скористайтеся тим, що значення часткової суми ряду $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$ при підсумовуванні досить великої кількості членів наближається до значення $\frac{\pi}{4}$. Рекомендований вигляд екрану:.

Задайте точність обчислення ПІ ->0.00001
*Значення числа ПІ з відповідною точністюдорівнює*3.143589
Підсумовано 502 члена ряду.

10. Написати програму, яка обчислює найбільший спільний дільник двох цілих чисел.

Додаток Д

Завдання до виконання лабораторної роботи № 5

1. Написати функцію Procent, яка повертає відсоток від отриманого в якості аргументу числа.

2. Написати функцію "Факторіал" та програму, яка використовує дану функцію для виведення таблиці факторіалів.

3. Написати функцію Dohod, яка обчислює дохід за вкладом. Вихідними даними для функції є: величина вкладу, процентна ставка (річних) і термін вкладу (кількість днів).

4. Написати функцію glasn, яка повертає 1, якщо символ, отриманий функцією як аргумент, є голосною буквою алфавіту, і нуль - в іншому випадку.

5. Написати функцію sogl, яка повертає 1, якщо символ, отриманий функцією як аргумент, є приголосною буквою алфавіту, і 0 - в іншому випадку.

6. Написати функцію frame, яка виводить на екран рамку. Як параметри функції повинні передаватися координати лівого верхнього кута і розмір рамки.

7. Написати функцію, яка обчислює значення a^b. Числа a й b можуть бути будь-якими дробовими позитивними числами.

8. Написати функцію, що забезпечує розв’язання квадратного рівняння. Параметрами функції повинні бути коефіцієнти і корені рівняння. Значення, які обраховує функція, мають передаватися в програму, яка викликається з інформацією про наявність коренів рівняння: 2 - два різних корені, 1 - корені однакові, рівняння не має розв’язків. Крім того, функція повинна перевіряти коректність вихідних даних. якщо вихідні дані невірні, то функція повинна повертати - 1.

9. Написати функцію, яка виводить рядок, що складається з однакових символів. Довжина рядка і символ є параметрами процедури.

10. Написати функцію, яка обчислює об’єм і площу поверхні паралелепіпеда.