Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Конструювання прототипів і шаблонів вебсторінок. Практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Технології друкованих і електронних видань»
спеціальності 186 Видавництво та поліграфія

Укладач: Р. Л. Тріщук

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

УДК 004.43 (076)
Т20

Укладач *Тріщук Руслан Любомирович*, канд. техн. наук, доц.

Рецензент *Зенкін М. А.*, докт. техн. наук, проф.
професор кафедри МАПВ КПІ ім. Ігоря Сікорського

Відповідальний редактор *Палюх О. О.*, д-р. техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 7 від 09.05.2024 р.)
за поданням вченої ради навчально-наукового видавничо-поліграфічного інституту
(протокол № 9 від 29.04.2024 р.)

Т20 **Конструювання прототипів і шаблонів вебсторінок. Практикум** [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Технології друкованих і електронних видань» спец. 186 «Видавництво та поліграфія» / КПІ ім. Ігоря Сікорського ; уклад.: Р. Л. Тріщук. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 70 с.

Навчальний посібник відповідає силабусу дисципліни «Конструювання прототипів і шаблонів вебсторінок» спеціальності 186 «Видавництво та поліграфія» освітньої програми «Технології друкованих і електронних видань» підготовки студентів Навчально-наукового видавничо-поліграфічного інституту. Наведено перелік практичних індивідуальних завдань, які ґрунтуються на застосуванні знань та навичок, отриманих під час вивчення теоретичного матеріалу.
Навчальний посібник призначений для студентів НН ВПІ КПІ ім. Ігоря Сікорського спеціальності 186 Видавництво та поліграфія. Також буде корисним студентам інших ЗВО, які готують фахівців за спеціальністю 186.

УДК 004.43 (076)

Реєстр. № НП 23/24-530. Обсяг 2,1 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП……..…… 4
1. МЕТА, ЗАВДАННЯ І ТЕМАТИКА РОБІТ ПРАКТИКУМУ..... 6
2. ОСНОВНІ ВИМОГИ ДО ВИКОНАННЯ РОБІТ ПРАКТИКУМУ..... 8
3. ЗМІСТ ТА ПЕРЕЛІК ПРАКТИЧНИХ РОБІТ..... 10
3.1. Вибір, встановлення та налаштування робочого середовища для роботи з JavaScript, HTML, CSS..... 10
3.2. Основи JavaScript: базові поняття; підключення коду до сторінки; змінні та строгий режим у JS; типи змінних і правила їх назв..... 17
3.3. Базовий синтаксис JavaScript: огляд типів даних, змінні та базові оператори; пошук і виправлення помилок у коді..... 22
3.4. Основи JavaScript: Оператори інкремент і декремент; модальні вікна, взаємодія з користувачем за допомогою alert, prompt, confirm; прості об'єкти та їх властивості..... 30
3.5. Основи JavaScript: умови, логічні оператори, цикли..... 38
3.6. Основи JavaScript: функції; callback-функції; методи та властивості рядків і чисел..... 43
3.7. Основи JavaScript: методи об’єкта; методи (функції) для роботи з масивами; деструктуризація об’єктів..... 50
3.8. Основи JavaScript: отримання елементів з вебсторінки; дії з елементами на вебсторінці..... 59
РЕКОМЕНДОВАНА ЛІТЕРАТУРА..... 68
ДОДАТОК А. Приклад титульного аркушу до практичних завдань..... 70

ВСТУП

Дисципліна «Конструювання прототипів і шаблонів вебсторінок» із циклу вибіркових компонентів освітньої програми (ОП) «Технології друкованих і електронних видань» підготовки студентів першого бакалаврського рівня вищої освіти Навчально-наукового видавничо-поліграфічного інституту спеціальності 186 Видавництво та поліграфія покликана поглибити знання, отримані студентами під час вивчення нормативної складової частин ОП.

Практикум з дисципліни «Конструювання прототипів і шаблонів вебсторінок» є важливим складником навчального процесу підготовки фахівців, логічним доповненням до лекційних і лабораторних занять та однією з базових ланок у системі їх практичної підготовки до створення вебсайтів, вебзастосунків тощо.

Навчальний посібник «Конструювання прототипів і шаблонів вебсторінок. Практикум» призначений для студентів денної та заочної форм навчання технічних спеціальностей, які цікавляться вебдизайном і веброзробкою. Посібник сприятиме розвитку креативного мислення, навичок конструювання прототипів та шаблонів вебсторінок, а також навичок практичного застосування інструментів для створення вебсторінок.

Курс відповідає нагальній ринковій потребі підготовки сучасних фахівців у сфері веброзробки та формує в них такі здатності:

- Здатність вчитися та оволодівати сучасними знаннями.
- Здатність застосовувати знання у практичних ситуаціях.
- Здатність приймати обґрунтовані рішення стосовно процесів, притаманних усім етапам виробництва друкованих й електронних видань, паковань, мультимедійних інформаційних продуктів та інших видів виробів видавництва та поліграфії.
- Здатність проєктувати структуру, конструкцію і дизайн друкованих та електронних видань, паковань, мультимедійних інформаційних продуктів та інших видів виробів видавництва й поліграфії, використовуючи сучасне

програмне та апаратне забезпечення, з урахуванням вимог до результату, наявних ресурсів та обмежень.

- Забезпечувати якість друкованих та електронних видань, паковань, мультимедійних інформаційних продуктів й інших видів виробів видавництва та поліграфії.
- Опрацьовувати текстову, графічну та мультимедійну інформацію з використанням сучасних інформаційних технологій і спеціалізованого програмного забезпечення.

1. МЕТА, ЗАВДАННЯ І ТЕМАТИКА РОБІТ ПРАКТИКУМУ

Мета робіт практикуму полягає в закріпленні знань, одержаних студентами під час вивчення дисципліни «Конструювання прототипів і шаблонів вебсторінок», застосуванні отриманих знань для вирішення конкретних завдань, сприянні самостійності під час аналізу та прийняття важливих професійних рішень, які б підвищили рівень підготовки.

Метою дисципліни є вивчення основних понять, принципів і технологій створення вебсайтів, основ серверного влаштування систем, способів розміщення сайтів на хостингу, підключення багатокомпонентних систем (код, база даних, поштові сервери, відображення, системи управління кодом), вивчення основних структурних блоків веб-порталів, створення шаблонних сторінок засобами проєктування мокапів (вайрфреймів) з елементами прототипування, відтворення запроєктованих сторінок засобами високорівневого програмування в середовищі CMS, налаштування шаблонів блогу/категорії/блог-посту, налаштування шаблонів магазину/категорії/продукту, відтворення сторінок із використанням попередньо створених шаблонів.

Метою робіт практикуму з дисципліни «Конструювання прототипів і шаблонів вебсторінок» є надати студентам можливість оволодіти практичними навичками роботи з мовою програмування JavaScript, необхідними для створення інтерактивних і динамічних вебсторінок.

Під час практикуму студенти навчаються:

- Розумінню основних концепцій та принципів мови JavaScript, включно з роботою зі змінними, типами даних, операторами, умовними конструкціями та циклами.
- Вивченню та застосуванню функцій, масивів та об'єктів для створення більш складних сценаріїв і вебдодатків.
- Розробці та відлагодженню скриптів, з використанням сучасних інструментів, таких як консоль браузера, відлагоджувачі та тестувальні фреймворки.

- Застосуванню найкращих практик і патернів проєктування для створення ефективного, надійного та масштабованого коду.
- Розробці інтерактивних та адаптивних вебсторінок, що відповідають сучасним стандартам і вимогам користувачів.
- За результатами практикуму студенти будуть здатні самостійно створювати вебсторінки з використанням JavaScript, а також розуміти та застосовувати основні концепції та принципи цієї мови програмування для розробки більш складних вебдодатків.

Роботи комп’ютерного практикуму виконують студенти першого (бакалаврського) рівня вищої освіти. Зміст і структура навчального посібника відображає сучасні тенденції розвитку видавничо-поліграфічної справи і забезпечує вирішення завдань щодо створення інтерактивних і динамічних вебсторінок.

2. ОСНОВНІ ВИМОГИ ДО ВИКОНАННЯ РОБІТ ПРАКТИКУМУ

Роботи практикуму з дисципліни "Конструювання прототипів та шаблонів вебсторінок" містять відповідні завдання, спрямовані на оволодіння студентами практичними навичками проєктування, створення та тестування вебсторінок. Для успішного виконання робіт практикуму необхідно дотримуватися наведених нижче вимог і правил. Роботи, виконані без дотримання цих вимог, можуть бути повернені студенту для доопрацювання.

Протокол роботи практикуму оформлюють у вигляді сторінок формату А4, дотримуючись вимог ДСТУ-3008-2015. Типова структура робіт практикуму є такою: титульний аркуш (додаток А); аркуш завдання; основна частина, додатки (за необхідності).

Звіт роботи виконують на аркушах А4, у протоколі стисло відображають хід роботи, отримані результати та висновки. Зміст виконаної роботи ілюструють на електронних носіях та, за можливості, виконаними проєктами (макети сторінок, шаблони, прототипи, сценарії тощо). Він повинен містити вступ, основну частину, висновки та список використаної літератури.

Вступ містить актуальність, мету, завдання та об'єкт дослідження. В основній частині детально описують виконані роботи, включно з аналізом вихідних даних, вибором методів, розробкою прототипів, тестуванням та оцінюванням результатів. У висновках подають підсумки роботи, отримані результати, формулюють рекомендації та перспективи подальшої роботи.

Список використаної літератури складають відповідно до ДСТУ 8302:2015 "Інформація та документація. Бібліографічний опис. Загальні вимоги та правила складання".

Під час виконання робіт практикуму необхідно дотримуватися таких вимог:

- Виконувати роботи самостійно, без сторонньої допомоги.
- Дотримуватися встановлених термінів виконання робіт.

- Використовувати лише надійні та перевірені джерела інформації.
- Виконувати роботи відповідно до вимог завдання та критеріїв оцінювання.
- Використовувати сучасні технології, інструменти та методики проєктування та створення вебсторінок.
- Дотримуватися вимог щодо якості та надійності вебсторінок, швидкості завантаження, сумісності з різними браузерами і пристроями, легкості використання та доступності.
- Проводити тестування та відлагодження вебсторінок, виправляти виявлені помилки та недоліки.
- Виконувати роботи в установленому порядку і забезпечувати їх своєчасне та якісне виконання.

Під час захисту робіт практикуму студент повинен:

- Знати зміст виконаної роботи, мету, завдання, об'єкт дослідження, методи, результати та висновки.
- Аргументовано відповідати на запитання, пов’язані з виконаною роботою, висловлювати власну думку та обґрунтовувати свої рішення.
- Презентувати виконану роботу, використовуючи сучасні презентаційні технології та засоби.
- Уміти захищати свою точку зору, аргументуючи її на основі надійних джерел інформації та власних досліджень.

3. ЗМІСТ ТА ПЕРЕЛІК ПРАКТИЧНИХ РОБІТ

3.1 ПРАКТИЧНА РОБОТА № 1

Вибір, встановлення та налаштування робочого середовища для роботи з JavaScript, HTML, CSS

Загальні теоретичні відомості

Мова сценаріїв JavaScript: JavaScript – це мова програмування, яку використовують для створення вебсторінок і вебдодатків. Вона дозволяє створювати динамічний вміст, обробляти користувацький ввід, створювати анімації та інші візуальні ефекти. JavaScript є невід'ємною частиною сучасних вебтехнологій та широко використовується в розробці вебзастосунків.

Робоче середовище – це програмне забезпечення, яке використовують для створення та редагування вебсторінок. Воно може включати текстовий редактор, інструменти для перевірки коду, налаштування середовища розробки тощо. Для роботи з JavaScript, HTML, CSS необхідно вибрати та встановити відповідне робоче середовище. Для розробки вебзастосунків популярними робочими середовищами є:

- Visual Studio Code (VS Code);
- Sublime Text;
- Atom;
- WebStorm;
- Brackets.

VS Code (Visual Studio Code) є безкоштовним і «легковагим» текстовим редактором, розробленим компанією Microsoft. Він підтримує велику кількість мов програмування, включно з JavaScript, HTML, CSS, та має багато корисних функцій для розробки вебзастосунків.

Особливості VS Code:

- Легка установка та налаштування: VS Code має простий і зрозумілий інтерфейс, який дозволяє легко встановити та налаштувати середовище розробки.
- Підтримка великої кількості мов програмування: VS Code підтримує більше ніж 30 мов програмування, включно з JavaScript, HTML, CSS, Python, Java, C++, PHP тощо.
- Вбудований термінал: VS Code має вбудований термінал, який дозволяє виконувати команди безпосередньо з редактора.
- Вбудований дебагер: VS Code має вбудований дебагер, який допомагає знаходити та виправляти помилки в коді.
- Git-інтеграція: VS Code має вбудовану підтримку Git, яка дозволяє виконувати команди Git безпосередньо з редактора.
- Велика кількість корисних розширень: VS Code має велику кількість розширень, які дозволяють розширити функціональність редактора та покращити продуктивність.
- Легке налаштування інтерфейсу користувача: VS Code дозволяє легко налаштувати інтерфейс користувача – тему, шрифт, розмір, кольори тощо.
- Швидка робота та низьке споживання ресурсів: VS Code працює швидко та споживає мало ресурсів, що робить його ідеальним вибором для розробки на ноутбуках і малопотужних комп'ютерах.

Переваги застосування VS Code:

- Легка установка та налаштування.
- Підтримка великої кількості мов програмування.
- Вбудований термінал.
- Вбудований дебагер.
- Git-інтеграція.

- Велика кількість корисних розширень.
- Легка настройка інтерфейсу користувача.
- Швидка робота та низьке споживання ресурсів.
- Безкоштовний та відкритий вихідний код.

Необхідні плагіни для VS Code:

- Node.js® – середовище виконання JavaScript, необхідне для роботи з пакетами файлів npm.
- All Autocomplete – автозавершення коду для JavaScript, HTML, CSS та інших мов програмування.
- Auto Close Tag – автоматичне закриття HTML-тегів.
- Auto Complete Tag – автозавершення HTML-тегів.
- Auto Rename Tag – автоматичне перейменування парних HTML-тегів.
- Beautify – форматування коду для JavaScript, HTML, CSS та інших мов програмування.
- Code Runner – виконання коду безпосередньо в VS Code.
- Import Cost – відображення розміру пакета npm при імпорті модулів.
- JavaScript (ES6) code snippets – готові фрагменти коду JavaScript.
- jshint – статичний аналізатор коду JavaScript.
- Live Server – запуск локального сервера для перегляду вебсторінок.
- Multiple clipboards for VSCode – збереження та відновлення фрагментів коду.
- Reactjs code snippets – готові фрагменти коду React.
- Sass – підтримка препроцесора Sass.
- ESLint – статичний аналізатор коду JavaScript, який допомагає виявити та виправити помилки в коді.

Ці плагіни допоможуть підвищити продуктивність і покращити якість коду при розробці вебсторінок, використовуючи JavaScript, та виявляти поширені помилки коду.

Загальна інструкція з установлення розширень у Visual Studio Code для роботи з JavaScript, HTML та CSS:

Встановлення Node.js®:

1. Для встановлення Node.js®, перейдіть на офіційний сайт Node.js® (https://nodejs.org/en/) та завантажте останню версію.
2. Після завантаження запустіть встановлювач та дотримуйтесь інструкцій, щоб встановити Node.js® на ваш комп'ютер.
3. Потім перезапустіть Visual Studio Code, щоб Node.js® почав працювати.

Встановлення інших (рекомендованих) розширень:

1. Запустіть Visual Studio Code.
2. Натисніть на іконку "Розширення" у лівому боковому меню або натисніть комбінацію клавіш "Ctrl+Shift+X".
3. У полі пошуку введіть назву розширення, яке ви хочете встановити. Наприклад, "JavaScript (ES6) code snippets".
4. Знайдіть розширення в списку та натисніть на нього.
5. Натисніть кнопку "Install", щоб встановити розширення.
6. Після завершення встановлення, перезапустіть Visual Studio Code, щоб розширення почало працювати.

Налаштування розширення jshint:

Для коректної роботи розширення JSHint (статичного аналізатора коду JavaScript) рекомендується створити файл *.jshintrc* з таким вмістом:

```
{
"camelcase" : true,
"indent": 2,
"undef": true,
"quotmark": false,
"maxlen": 120,
"trailing": true,
"curly": true,
"strict": false,
"browser": true,
"devel": true,
"jquery": true,
"esversion": 9,
```

```
"node": true
}
```

та зберегти його в кореневу папку розташування майбутніх проєктів.

Файл *.jshintrc* використовують для конфігурації інструменту JSHint, який є статичним аналізатором коду JavaScript. Цей інструмент допомагає виявляти помилки та неточності в коді, а також забезпечує дотримання певних стандартів кодування.

Файл *.jshintrc* містить набір параметрів, які визначають, як JSHint буде аналізувати код. У прикладі, наведеному вище, встановлено значення по замовчуванню для таких параметрів:

"*camelcase*" – вимагає використання camelCase для назв змінних.

"*indent*" – встановлює кількість пробілів для відступу.

"*undef*" – вимагає оголошення всіх змінних перед використанням.

"*quotmark*" – встановлює тип лапок, які повинні використовуватися в коді.

"*maxlen*" – встановлює максимальну довжину рядка коду.

"*trailing*" – вимагає або забороняє використання пробілів у кінці рядка.

"*curly*" – вимагає використання фігурних дужок для блоків коду.

"*strict*" – вимагає використання суворого режиму JavaScript.

"*browser*" – вказує, що код призначений для використання у веббраузері.

"*devel*" – дозволяє використання функцій, призначених для розробки (наприклад, console.log).

"*jquery*" – вказує, що код використовує бібліотеку jQuery.

"*esversion*" – встановлює версію ECMAScript, яка повинна використовуватися в коді.

"*node*" – вказує, що код призначений для використання в середовищі Node.js.

Загалом, створення файлу .jshintrc дозволяє налаштувати JSHint відповідно до власних потреб і стандартів кодування, що допомагає забезпечити «чистий», надійний та легко зрозумілий код.

Мета роботи – навчитися вибирати, встановлювати та налаштовувати робоче середовище для роботи з JavaScript, HTML, CSS.

Завдання:

1. Здійснити вибір та встановити робоче середовище (VS code чи ін.) для роботи з JavaScript, HTML, CSS.
2. Виконати налаштування (встановлення мовного пакету, розширень тощо).
3. Обґрунтувати свій вибір у формі звіту (у довільній формі).
4. Завантажити файл звіту (назва файлу повинна містити групу і прізвище студента транслітом).

Хід роботи:

1. Вибір робочого середовища:
   - Ознайомитися з оглядом популярних робочих середовищ для розробки вебсторінок.
   - Вибрати одне з них (рекомендовано: VS code) та обґрунтувати свій вибір.
2. Встановлення робочого середовища:
   - Завантажити та встановити обране робоче середовище.
   - Переконатися, що воно працює коректно.
3. Налаштування робочого середовища:
   - Встановити мовний пакет (за потреби).
   - Встановити необхідні розширення для полегшення роботи (наприклад, для форматування коду, автодоповнення тощо).
   - Налаштувати зовнішній вигляд та функції робочого середовища відповідно до своїх потреб.
4. Обґрунтування вибору:
   - Скласти звіт, в якому обґрунтувати свій вибір робочого середовища та налаштувань.

– Завантажити файл звіту на відповідну платформу (назва файлу повинна містити групу та прізвище студента транслітом).

Контрольні запитання:

1. Що таке робоче середовище?
2. Які функції може виконувати робоче середовище для розробки вебсторінок?
3. Які робочі середовища є нині популярними для розробки вебсторінок?
4. Як встановити мовний пакет у вибраному робочому середовищі?
5. Що таке розширення для робочого середовища?
6. Як налаштувати зовнішній вигляд і функції робочого середовища відповідно до своїх потреб?
7. Як обґрунтувати свій вибір робочого середовища та налаштувань у звіті?

3.2 ПРАКТИЧНА РОБОТА № 2

Основи JavaScript: базові поняття; підключення коду до сторінки; змінні та строгий режим у JS; типи змінних і правила їх назв

Загальні теоретичні відомості:

JavaScript – це мова програмування, яку використовують для створення вебсторінок і вебдодатків. Вона дозволяє створювати динамічний вміст, обробляти користувацький ввід, створювати анімації та інші візуальні ефекти. JavaScript як невід'ємну частину сучасних вебтехнологій широко використовують у розробці вебзастосунків.

Базові поняття JavaScript:

- Скрипт – програма, написана мовою JavaScript, яка виконується веббраузером.
- Підключення скрипту до сторінки – процес додавання файлу JavaScript до HTML-сторінки за допомогою тегу <script>.
- Змінні – іменовані області пам'яті, які використовують для зберігання даних.
- Строгий режим – режим JavaScript, який забезпечує більш сувору перевірку коду та допомагає уникнути деяких поширених помилок.
- Типи змінних – різні типи даних, які можуть зберігатися у змінних, такі як числа, рядки, булеві значення, об'єкти та масиви.
- Правила назв – правила, що визначають, як можна називати змінні, функції та інші ідентифікатори в JavaScript.

Мета роботи – ознайомлення з базовими поняттями мови програмування JavaScript, підключенням коду до сторінки, змінними та строгим режимом у JS, типами змінних і правилами створення їх назв.

Завдання:

1. Налаштуйте робочий простір для JavaScript.
2. Створіть HTML-сторінку з базовою розміткою та збережіть її у файл *index.html*.
1. У поточній директорії (або у вкладеній) створіть файл з розширенням .js.
2. Приєднайте створений файл скрипту до сторінки *index.html*.
3. Напишіть код, обравши вихідні дані (табл. 3.2.1) і тип завдання (табл. 3.2.2) згідно зі своїм варіантом (номером у списку групи):
4. Значення змінної result виведіть у консоль чи модальне вікно.
5. У кінці скрипту «закоментуйте» дані про себе та виконану вами роботу.
6. Збережіть створені файли та, не змінюючи структуру їх розташування, помістіть в архів із назвою PR-2-[група]-[прізвище транслітом].
7. Вивантажте архів у відповідну папку в сховищі, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Таблиця 3.2.1 – Вихідні дані за варіантами

№ варіанта	**Вид завдання:**	**Значення**		
		X	**Y**	**Z**
1	A	5 см	17 см	–
2	D	45 см	3 м	–
3	B	30 см	40 см	0,5 м
4	C	7 м	–	–
5	E	93 см	14 м	
6	D	6 м	87 см	–
7	A	2 м	48 см	–
8	B	7 см	24 см	0,25 м
9	C	12 см	–	–
10	E	64 м	73 м	–
11	D	35 см	74 см	–
12	C	31 см	–	–
13	A	29 см	5 м	–
14	E	2 м	14 см	–
15	B	21 см	10 см	29 см
16	D	5 см	59 см	–
17	A	62 см	9 см	–
18	E	2 м	34 см	–
19	C	24 см	–	–

20	B	8 см	17 см	15 см
21	E	5 см	98 см	–
22	D	14 см	19 см	–
23	A	8 м	94 см	–
24	C	16 м	–	–
25	B	9 см	0,12 м	0,15 м

Таблиця 3.2.2 – Види завдань

Вид завдання	**Завдання**
A	Напишіть скрипт, який знаходить площу прямокутника висотою ***X*** (числова змінна *height*), шириною ***Y*** (числова змінна *width*). Числове значення отриманої площі збережіть у змінну s. Результат помістіть у змінну *result* у вигляді: «Площа прямокутника зі сторонами (значення *height*) см та (значення *width*) см становить: (значення *s*) кв. см.»
B	Напишіть скрипт, який знаходить периметр трикутника зі сторонами ***X*** (числова змінна side1), ***Y*** (числова змінна side2) та ***Z*** (числова змінна side3). Числове значення отриманого периметра збережіть у змінну *p*. Результат помістіть у змінну *result* у вигляді: «Периметр трикутника зі сторонами: (значення *side1*) см, (значення *side2*) см та (значення *side3*) см становить – (значення *p*) см».
C	Напишіть скрипт, який знаходить площу круга діаметром ***X*** (числова змінна *dCircle*). Числове значення отриманої площі збережіть у змінну *s*. (Оператор зведення у степінь: **) . Результат помістіть у змінну result у вигляді: «Площа круга з діаметром (значення *dCircle*) см становить: (значення *s*) кв. см. »
D	Напишіть скрипт, який знаходить периметр прямокутника висотою ***X*** (числова змінна *height*), шириною ***Y*** (числова змінна *width*). Числове значення отриманого периметра збережіть у змінну *p*. Результат помістіть у змінну *result* у вигляді: «Периметр прямокутника зі сторонами (значення *height*) см та (значення *width*) см становить: (значення *p*) см».
E	Напишіть скрипт, який знаходить об'єм циліндра заввишки ***X*** (числова змінна *heightC*) та діаметром основи ***Y*** (числова змінна *dC*). Числове значення отриманого об'єму помістіть у змінну v. Результат помістіть у змінну result у вигляді: «Об'єм циліндра заввишки (значення *heightC*) м та діаметром основи (значення *dC*) м становить: (значення *v*)».

Хід роботи:

1. Налаштуйте робочий простір для JavaScript. Для цього встановіть Visual Studio Code та необхідні розширення, як описано в попередній практичній роботі.

2. Створіть HTML-сторінку з базовою розміткою. Для цього створіть файл *index.html* у поточній директорії та додайте такий код:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title> Практична робота №2</title>
</head>
<body>
   <h1>Практична робота №2</h1>
   <!-- Тут буде виводитись результат роботи скрипту -->
</body>
</html>
```

3. У поточній директорії (або у вкладеній) створіть файл з розширенням .js. Для цього натисніть праву кнопку миші на поточній директорії та виберіть "Створити новий файл". Назвіть файл *script.js*.

4. Приєднайте створений файл скрипту до сторінки *index.html*. Для цього додайте такий код у тег <body> файлу *index.html*:

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

5. Напишіть скрипт, який виконує завдання згідно з вашим варіантом (табл. 3.2.1 та табл. 3.2.2). Наприклад, знаходить об'єм циліндра заввишки 4 м і діаметром основи 2 м. Для цього додайте такий код у файл *script.js*:

```
// Оголошення змінних
const hCylinder = 4;
const dCylinder = 2;
const rCylinder = dCylinder / 2;
const PI = 3.14;

// Обчислення площі основи циліндра (s)
let sCylinder = dCylinder / 2 ** 2 * PI;
```

```
// Обчислення об'єму циліндра
let vCylinder = sCylinder * hCylinder;

// Формування результату
let result = `Об'єм циліндра заввишки ${heightC} м та діаметром основи ${dC} м становить: ${v}`;

// Виведення результату
console.log(result);
```

6. У кінці скрипту «закоментуйте» дані про себе та виконану вами роботу. Для цього додайте такий код у файл *script.js*:

```
// Практ. роб. № 2. Виконав студент групи СТ-11 Петренко Володимир.
```

7. Збережіть створені файли та помістіть в архів із назвою PR-2-[група]-[прізвище транслітом].

Контрольні запитання:

1. Що таке JavaScript?
2. Які базові поняття JavaScript вам відомі?
3. Як підключити файл JavaScript до HTML-сторінки?
4. Що таке змінні в JavaScript?
5. Що таке строгий режим у JavaScript?
6. Які типи змінних існують у JavaScript?
7. Якими є правила назв для змінних у JavaScript?
8. Як виконувати математичні обчислення за допомогою JavaScript?
9. Що таке «коментування» коду в JavaScript?
10. Як вивести результат роботи скрипту в консоль чи модальне вікно?

3.3 ПРАКТИЧНА РОБОТА № 3

Базовий синтаксис JavaScript: огляд типів даних, змінні та базові оператори; пошук та виправлення помилок у коді

Загальні теоретичні відомості:

Базовий синтаксис JavaScript включає в себе такі поняття, як типи даних, змінні та оператори.

У JavaScript дані поділяються на два основні типи: примітивні (прості типи) та об'єктні (звичайні об’єкти та спеціальні об’єкти). До примітивного типу даних належать числа, рядки, булеві значення, *null* та *undefined*. Об'єктні типи даних включають в себе об'єкти, масиви та функції (рис. 3.3.1).

Рис. 3.3.1. Типи даних у JavaScript

Змінні в JavaScript використовуються для зберігання даних, які можуть змінюватися під час виконання програми. Для створення змінної в JavaScript використовується ключове слово *let* або *const*. *Let* слугує для створення змінної, значення якої може бути модифіковане, а *const* застосовується для створення змінної, значення якої не може бути змінене.

Оператори в JavaScript призначені для виконання різних операцій, таких як математичні обчислення, порівняння, логічні операції та інші. Існує кілька типів операторів у JavaScript:

- Арифметичні оператори: +, -, *, /, % (залишок від ділення) та ** (піднесення до степеня).
- Оператори порівняння: ==, !=, ===, !==, >, <, >=, <=.

- Логічні оператори: && (і), || (або), *!* (не).
- Оператори присвоєння: =, +=, -=, *=, /=, %=, **=.
- Оператори інкременту та декременту: ++, --.
- Умовні оператори: *if, else if, else, switch*.
- Оператори циклу: *for, while, do-while*.
- Оператори роботи з масивами: *push, pop, shift, unshift, splice, slice, sort, forEach, map, filter, reduce*.
- Оператори роботи з об'єктами: . (доступ до властивостей), [] (доступ до властивостей за індексом), *for-in* (цикл по властивостях об'єкта), *Object.keys(), Object.values(), Object.entries(), Object.assign()*.

Приклади використання змінних та операторів у JavaScript:

```
let a = 10;
let b = 20;
let c = a + b; // c = 30
let d = a * b; // d = 200
let e = a / b; // e = 0.5
let f = a % b; // f = 10
let g = a ** b; // g = 100000000000000000000
let h = a == b; // h = false
let i = a != b; // i = true
let j = a === b; // j = false
let k = a !== b; // k = true
let l = a > b; // l = false
let m = a < b; // m = true
let n = a >= b; // n = false
let o = a <= b; // o = true
let p = a && b; // p = 20
let q = a || b; // q = 10
let r = !a; // r = false
let s = a + " рядок"; // s = "10 рядок"
let t = a * "5"; // t = 50
let u = a / "2"; // u = 5
let v = a % "3"; // v = 1
let w = a ** "2"; // w = 100
let x = a++; // x = 10, a = 11
let y = --b; // y = 19, b = 19
let z = { name: "John", age: 30 }; // об'єкт
let arr = [1, 2, 3, 4, 5]; // масив
let arr2 = ["apple", "banana", "orange"]; // масив
```

Ці приклади демонструють, як використовувати змінні та оператори в JavaScript для виконання деяких простих операцій.

Пошук і виправлення помилок у коді є важливою частиною розробки програмного забезпечення. Хибні виконання коду можуть виникати через синтаксичні та логічні помилки, неправильне введення даних та з інших причин. Для виправлення помилок необхідно використовувати навички дебагування, такі як перевірка синтаксису, перевірка значень змінних, використання відладчика тощо.

Мета роботи – ознайомитися з базовим синтаксисом JavaScript, зокрема з типами даних, змінними та базовими операторами, а також навчитися шукати та виправляти помилки в коді.

Завдання:

1. Створіть HTML-сторінку з базовою розміткою та збережіть її у файл *index.html*.
2. У поточній директорії (або у вкладеній) створіть файл з довільним ім’ям і розширенням .js (наприклад: *my.js*).
3. Приєднайте створений файл скрипту до сторінки *index.html*.
4. У створений файл додайте код відповідно до свого варіанту (подано нижче) і збережіть.

Варіант 1:

```
"use strict"

/* Розрахунок периметра прямокутника*/
let aRectangle = 35 /* Сторона прямокутника (a)*/
let bRectangle = 21 /* Сторона прямокутника (b)*/
let pRectangle /* Периметр прямокутника (p)*/
PRectangle = aRectangle * bRectangle;
let resultRectangle
resultRectangle = `Периметр прямокутника зі сторонами a=${aRectangle} см та b=${bRectangle} см – становить: ${pRectangle} кв. см.`;

console.log(“resultRectangle”)
```

Варіант 2:

```
"use strict"

/* Розрахунок площі круга*/
let dCircle = 11; /* Діаметр круга (d)*/
const sCircle; /* Площа круга (s)*/
sCircle = dCircle/2 ** 2 * 3.14;
let resultCircle
ResultCircle = "Площа круга з діаметром d = " + dCircle + " см – становить: " + sCircle + " кв. см."

console.log(resultCircle)
```

Варіант 3:

```
"use strict"

/* Розрахунок об'єму циліндра*/
let dCylinder = "15" /* Діаметр основи циліндра (d)*/
let hCylinder = "21" /* Висота циліндра (h)*/
const sCylinder;
sCylinder = dCylinder / 2 ** 2 * 3.14; /* Площа основи циліндра (s)*/
const vCylinder; /* Об'єм циліндра (v)*/
vCylinder = sCylinder * hCylinder;

const resultCylinder;
resultCylinder = `Об'єм циліндра з діаметром основи d = ` + `dCylinder` + ` та висотою ` + hCylinder + ` см – становить: ` + vCylinder + ` куб. см.`;

console.log(resultCylinder);
```

Варіант 4:

```
"use strict"

/* Розрахунок площі круга
let dCircle = 5; /* Діаметр круга (d)*/
sCircle; /* Площа круга (s)*/
sCircle = (dCircle/2) ** 2 * 3.14;
resultCircle;
resultCircle = "Площа круга з діаметром d = " + "dCircle" + " см – становить: " + "sCircle" + " кв. см.";

console.log(resultCircle);
```

Варіант 5:

```
"use strict"

/* Розрахунок площі прямокутника*/
let aRectangle = 35 /* Сторона прямокутника (a)*/
let bRectangle = 21 /* Сторона прямокутника (b)*/
let sRectangle /* Периметр прямокутника (p)*/
SRectangle = aRectangle * bRectangle;
let resultRectangle
resultRectangle = `Площа прямокутника зі сторонами a=${aRectangle} см та b=${bRectangle} см – становить: ${pRectangle} кв. см.`;

console.log(“resultRectangle”)
```

Варіант 6:

```
"use strict";

/* Розрахунок периметра прямокутника*/
aRectangle = "21"; /* Сторона прямокутника (a)*/
bRectangle = "7"; /* Сторона прямокутника (b)*/
let pRectangle; /* Периметр прямокутника (p)*/
pRectangle = aRectangle + bRectangle * 2
let resultRectangle;
ResultRectangle = `Периметр прямокутника зі сторонами a=${aRectangle} см та b=${bRectangle} см – становить: ${pRectangle} см.`;

console.log(resultRectangle)
```

Варіант 7:

```
use strict

/* Розрахунок об'єму циліндра
dCylinder = 15 /* Діаметр основи циліндра (d)*/
hCylinder = 21 /* Висота циліндра (h)*/
let sCylinder
sCylinder = `(dCylinder/2) ** 2 * 3.14` /* Площа основи циліндра (s)*/
let vCylinder /* Об'єм циліндра (v)*/
VCylinder = `sCylinder * hCylinder`

const resultCylinder;
resultCylinder = "Об'єм циліндра з діаметром основи d = " + dCylinder + " та висотою " + hCylinder + " см – становить: " + vCylinder + " куб. см.";

alert(resultCylinder);
```

Варіант 8:

```
"use strict"

/* Розрахунок периметра прямокутника
aRectangle = 9; /* Сторона прямокутника (a)*/
bRectangle = 5; /* Сторона прямокутника (b)*/
const pRectangle; /* Периметр прямокутника (p)*/
pRectangle = (aRectangle + bRectangle) ** 2,
const resultRectangle;
resultRectangle = `Периметр прямокутника зі сторонами a=${aRectangle} см та b=${bRectangle} см – становить: ${pRectangle} см.`;
console.log(resultRectangle);
```

Варіант 9:

```
"use strict",

/* Розрахунок об'єму циліндра*/
dCylinder = 15, /* Діаметр основи циліндра (d)*/
hCylinder = 21, /* Висота циліндра (h)*/
let sCylinder;
sCylinder = "(dCylinder/2) ** 2 * 3.14", /* Площа основи циліндра (s)*/
let vCylinder; /* Об'єм циліндра (v)
vCylinder = sCylinder * hCylinder,

let resultCylinder;
resultCylinder = "Об'єм циліндра з діаметром основи d = " + "dCylinder" + " та висотою " + hCylinder + " см – становить: " + "vCylinder" + " куб. см.";

alert(resultCylinder)
```

Варіант 10:

```
use strict

/* Розрахунок площі круга*/
let dCircle = `11`; /* Діаметр круга (d)*/
let sCircle; /* Площа круга (s)*/
sCircle = `(dCircle/2) ** 2 * 3.14`;
let resultCircle;
resultCircle = "Площа круга з діаметром d = " + dCircle +  см – становить:  + "sCircle" + " кв. см."

console.log("resultCircle");
```

5. Запропонований код містить помилки. Необхідно знайти всі помилки в коді та виправити їх, зберегти файл і протестувати його на правильність роботи.
6. У кінці скрипту «закоментуйте» дані про себе та виконану вами роботу.
7. Збережіть створені файли та, не змінюючи структуру їх розташування, помістіть в архів із назвою PR-3-[група]-[прізвище транслітом].
8. Вивантажте архів у відповідну папку сховища, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Хід роботи:

1. Створіть HTML-сторінку з базовою розміткою. Для цього створіть файл index.html у поточній директорії та додайте такий код:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Практична робота №3</title>
</head>
<body>
   <h1>Практична робота №3</h1>
   <!-- Тут буде виводитись результат роботи скрипту -->
</body>
</html>
```

2. У поточній директорії (або у вкладеній) створіть файл з довільним ім’ям і розширенням .js (наприклад: my.js).

3. Приєднайте створений файл скрипту до сторінки index.html. Для цього додайте такий код у тег <body> файлу index.html:

```
<script src="my.js"></script>
```

4. У створений файл додайте код згідно свого варіанта та збережіть.

5. Запропонований код містить помилки. Знайдіть всі помилки в коді та виправте їх, збережіть файл та виконайте його тестування щодо правильності роботи.

6. У кінці скрипту «закоментуйте» дані про себе та виконану вами роботу.

Приклад:

```
// Практична робота №3.

// Варіант 7.

// Виконав студент групи МВ-71 Петренко Ігор.
```

7. Збережіть створені файли та, не змінюючи структури їх розташування, помістіть в архів із назвою PR-3-[група]-[прізвище транслітом].

8. Вивантажте архів у відповідну папку сховища, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Контрольні запитання:

1. Що таке JavaScript?
2. Які базові типи даних є в JavaScript?
3. Що таке змінні в JavaScript?
4. Які оператори існують у JavaScript?
5. Що таке помилка в коді та як її виправити?
6. Що таке дебагування та якими інструментами можна користуватися для дебагування JavaScript коду?
7. Як коментувати код у JavaScript?
8. Що таке строгий режим у JavaScript?
9. Як правильно підключати JavaScript-файл до HTML сторінки?
10. Як вивести результат роботи скрипту в консоль чи на сторінку?

3.4 ПРАКТИЧНА РОБОТА № 4

Основи JavaScript: Оператори інкремент і декремент; модальні вікна, взаємодія з користувачем за допомогою alert, prompt, confirm; прості об'єкти та їх властивості

Загальні теоретичні відомості:

Оператори інкремент і декремент у JavaScript використовують для збільшення або зменшення значення змінної на одиницю. Оператор інкремент позначають як ++, а оператор декремент – як --. Ці оператори можуть бути використані як префіксні (тобто перед змінною), так і постфіксні (після змінної).

Приклади використання операторів інкремент і декремент у JavaScript:

```
let x = 10;
let y = 20;

// Префіксний інкремент
++x; // x = 11
console.log(x); // Виведе: 11

// Постфіксний інкремент
y++; // y = 21
console.log(y); // Виведе: 21

// Префіксний декремент
--x; // x = 10
console.log(x); // Виведе: 10

// Постфіксний декремент
y--; // y = 20
console.log(y); // Виведе: 20
```

У цьому прикладі маємо дві змінні *x* та *y*, які містять значення 10 та 20 відповідно. Далі використовуємо оператори інкремент і декремент для збільшення або зменшення значення цих змінних на одиницю:

- Використовуємо префіксний інкремент *++x* для збільшення значення змінної x на одиницю до виконання операції. Таким чином, значення змінної *x* стає рівним 11.

- Використовуємо постфіксний інкремент y++ для збільшення значення змінної y на одиницю після виконання операції. Таким чином, значення змінної y стає рівним 21.
- Використовуємо префіксний декремент --x для зменшення значення змінної x на одиницю до виконання операції. Таким чином, значення змінної x стає рівним 10.
- Використовуємо постфіксний декремент y-- для зменшення значення змінної y на одиницю після виконання операції. Таким чином, значення змінної y стає рівним 20.

Оператори інкремент і декремент можуть бути використані не тільки з числовими змінними, а й з рядками, булевими значеннями та іншими типами даних.

Приклад використання операторів інкремент і декремент з рядками:

```
let str = "abc";

console.log(str++); // Виведе: "abc"
console.log(str); // Виведе: "abd"

console.log(++str); // Виведе: "abe"
console.log(str); // Виведе: "abe"
```

У цьому прикладі маємо змінну *str*, яка містить рядок "abc". Далі використовуємо оператори інкремент і декремент для зміни значення цієї змінної.

- Використовуємо постфіксний інкремент *str++* для збільшення значення змінної *str* на одиницю після виконання операції. Таким чином, значення змінної *str* дорівнює "*abd*".
- Використовуємо префіксний інкремент *++str* для збільшення значення змінної *str* на одиницю до виконання операції. Таким чином, значення змінної *str* стає рівним "*abe*".

Оператори інкремент і декремент можуть бути корисними для виконання простих операцій над змінними в JavaScript. Вони дозволяють зменшити кількість коду, необхідного для виконання цих операцій, та зробити його більш читабельним.

Модальні вікна в JavaScript – це вікна, які з'являються поверх основного вікна вебсторінки та вимагають від користувача взаємодії. Існує три основні типи модальних вікон у JavaScript: *alert, prompt* та *confirm.*

alert – модальне вікно, яке відображає повідомлення користувачеві та вимагає від нього підтвердження. Це вікно не має жодних інших кнопок, окрім кнопки "OK".

Синтаксис використання *alert* у JavaScript:

```
alert("Повідомлення для користувача");
```

prompt – модальне вікно, яке дозволяє користувачеві ввести деяке значення. Це вікно містить поле введення та дві кнопки: "OK" і "Скасувати".

Синтаксис використання prompt у JavaScript:

```
let userInput = prompt("Повідомлення для користувача", "Значення за замовчуванням");
```

У цьому прикладі ми використовуємо *prompt* для отримання від користувача деякого значення. Ми передаємо два аргументи: повідомлення для користувача та значення за замовчуванням. Значення, введене користувачем, зберігається в змінній *userInput*.

confirm – модальне вікно, яке відображає повідомлення користувачеві та вимагає від нього підтвердження. Це вікно містить дві кнопки: "OK" та "Скасувати".

Синтаксис використання *confirm* у JavaScript:

```
let userConfirmation = confirm("Повідомлення для користувача");
```

У цьому прикладі ми використовуємо *confirm* для отримання від користувача підтвердження та передаємо один аргумент: повідомлення для користувача. Якщо користувач натисне кнопку "*OK*", значення *true* буде збережене в змінній *userConfirmation*. Якщо натиснути кнопку "Скасувати", то значення *false* буде збережене в змінній *userConfirmation*.

Ці модальні вікна можна використовувати для отримання від користувача деякої інформації або її підтвердження, а також для відображення повідомлень. Однак вони можуть бути дратівливими для користувачів, якщо їх використовують надто часто або в невідповідний момент. Тому їх використанням не варто зловживати.

Об'єкти в JavaScript – це набори властивостей, які описують деякий об'єкт. Властивості об'єкта можуть бути різними типами даних, як-от: числа, рядки, булеві значення, об'єкти та масиви. Об'єкти в JavaScript створюють за допомогою літералу об'єкта {}.

Отже об'єкти в JavaScript є невпорядкованими колекціями властивостей, які можуть бути функціями, числами, рядками або будь-якими іншими типами даних. Вони дозволяють зберігати та передавати великі набори даних, а також створювати методи, які виконують дії на основі цих даних.

Синтаксис створення об'єкта в JavaScript:

```
let objectName = {
  property1: value1,
  property2: value2,
  property3: value3,
  ...
  propertyN: valueN
};
```

Де:

- *objectName* – назва об'єкта, яка може бути будь-якою допустимою назвою змінної.
- *propertyX* – властивості об'єкта, які можуть бути будь-якими допустимими назвами змінних.
- *valueX* – значення властивостей, які можуть бути будь-якими типами даних, включно з функціями.

Приклад створення та використання об'єкта:

```
// Створення об'єкта
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2020,
  color: "Blue",
  startEngine: function() {
    console.log("Engine started");
  }
};

// Доступ до властивостей об'єкта
```

```
console.log(car.brand); // Виведе: Toyota
console.log(car.model); // Виведе: Corolla

// Використання методу об'єкта
car.startEngine(); // Виведе: Engine started

// Зміна властивостей об'єкта
car.color = "Red";
console.log(car.color); // Виведе: Red
```

У цьому прикладі створено об'єкт car з властивостями *brand*, *model*, *year*, *color* і методом *startEngine*. Також продемонстровано, як отримати доступ до властивостей і методів об'єкта, а також як змінити значення властивостей.

Мета роботи – ознайомитися з операторами інкремент і декремент у JavaScript, модальними вікнами, взаємодією з користувачем за допомогою *alert*, *prompt*, *confirm*, простими об'єктами та їх властивостями. Набути практичних навичок роботи з цими конструкціями та елементами мови програмування.

Завдання:

1. Створити змінну *numberOfBooks* та в неї помістити відповідь від користувача на питання: "Скільки книг ви прочитали за рік?".
2. Створити об'єкт *personalBookDB* і в нього помістити такі властивості:

- *count* – сюди передається відповідь на перше питання (числовий тип даних);
- *books* – в цю властивість помістити порожній об'єкт;
- *actors* – теж помістити порожній об'єкт;
- *genres* – сюди помістити порожній масив;
- *privat* – в цю властивість помістити boolean (логічне) значення false.

3. Задайте користувачеві по два рази пару питань:

- "Одна з останніх прочитаних книг?".
- "На скільки оціните її (від 1 до 10)?". Відповіді варто помістити в окремі змінні. Записати відповіді в об'єкт *books* у форматі (приклад):

books: {

' Тіні забутих предків': 8

' Украдене щастя': 9

}

4. Перевірте, щоб все працювало без помилок у консолі.

5. У кінці скрипту «закоментуйте» дані про себе та виконану вами роботу.

6. Збережіть створені файли та помістіть в архів із назвою PR-2-[група]-[прізвище транслітом].

7. Вивантажте архів з виконаною роботою за відповідним посиланням.

Хід роботи:

1. Створіть HTML-сторінку з базовою розміткою. Для цього створіть файл index.html у поточній директорії та додайте такий код:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Практична робота №4</title>
</head>
<body>
   <h1>Практична робота №4</h1>
   <!-- Тут буде виводитись результат роботи скрипту -->
</body>
</html>
```

2. У поточній директорії (або у вкладеній) створіть файл з ім’ям: ваше прізвище (транслітом) і розширенням .js (наприклад: *petrenko.js*).

3. Приєднайте створений файл скрипту до сторінки *index.html*. Для цього додайте такий код у тег <body> файлу index.html:

```
<script src="petrenko.js"></script>
```

4. У створений файл додавайте код згідно із завданням.

5. Створіть змінну *numberOfBooks* і запитайте в користувача про кількість прочитаних ним книг:

```
let numberOfBooks = ...
```

6. Створіть об'єкт *personalBookDB* з необхідними властивостями:

```
let personalBookDB = {
  ... ,
  ... ,
  ... ,
  ... ,
  ...
};
```

7. Двічі запитайте в користувача назву та оцінку книги, запишіть їх в об'єкт *books*.
8. Перевірте роботу скрипту та виправте помилки (якщо є).
9. У кінці коду «закоментуйте» дані про себе та виконану вами роботу, приклад:

```
// Практ. роб. № 4. Виконала студентка групи СТ-11 Васильєва Наталія.
```

10. Збережіть створені файли та, не змінюючи структуру їх розташування, помістіть у архів із назвою PR-4-[група]-[прізвище транслітом].
11. Вивантажте архів у відповідну папку сховища, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Контрольні запитання:

1. Що таке оператори інкремент і декремент?
2. Яка різниця між префіксними та постфіксними операторами інкременту та декременту?
3. Наведіть приклади використання функцій alert(), prompt() і confirm().
4. Що таке об'єкт у JavaScript?
5. Як створити об'єкт за допомогою літералу об'єкта?
6. Як додати нову властивість в об'єкт?
7. Як отримати значення властивості об'єкта?
8. Як видалити властивість об'єкта?
9. Як перевірити, чи існує властивість в об'єкті?

10. Що таке масив у JavaScript?
11. Як створити масив?
12. Як додати елемент у масив?
13. Як отримати елемент масиву за індексом?
14. Як видалити елемент масиву?

3.5 ПРАКТИЧНА РОБОТА № 5

Основи JavaScript: умови, логічні оператори, цикли

Загальні теоретичні відомості:

1. Умови

Умовні конструкції в JavaScript дозволяють виконувати різний код залежно від певних умов. Для цього використовують оператори *if*, *if...else*, *if...else if...else* та *switch*.

Приклад:

```
let x = 5;
if (x > 0) {
  console.log("x додатнє");
} else if (x < 0) {
  console.log("x від'ємне");
} else {
  console.log("x дорівнює нулю");
}

let day = 3;
switch (day) {
  case 1:
    console.log("Понеділок");
    break;
  case 2:
    console.log("Вівторок");
    break;
  case 3:
    console.log("Середа");
    break;
  default:
    console.log("Інший день тижня");
}
```

2. Логічні оператори

Логічні оператори використовують для побудови складних умов. У JavaScript є три логічні оператори: && (логічне "І"), || (логічне "АБО") та *!* (логічне "НЕ").

Приклад:

```
let x = 5;
let y = 10;

if (x > 0 && y > 0) {
  console.log("x та y додатні");
}

if (x > 0 || y < 0) {
  console.log("x додатнє або y від'ємне");
}

if (!(x === y)) {
  console.log("x не дорівнює y");
}
```

3. Цикли

Цикли в JavaScript використовують для повторення певного коду декілька разів. У JavaScript існує три основні типи циклів: *for*, *while* та *do...while*.

Приклад:

```
// Цикл for
for (let i = 0; i < 5; i++) {
  console.log(i);
}

// Цикл while
let j = 0;
while (j < 5) {
  console.log(j);
  j++;
}

// Цикл do...while
let k = 0;
do {
  console.log(k);
  k++;
} while (k < 5);
```

Мета роботи – освоїти навички роботи з умовами, логічними операторами та циклами в JavaScript.

Завдання:

Використовуючи попередню роботу (практ. р. № 4) внести такі зміни в код (значення ***A***, ***B***, ***C***, ***D*** – за варіантами (табл. 3.5.1)):

Таблиця 3.5.1 – Вихідні дані за варіантами

№ варіанта	Значення			
	A	***B***	***C***	***D***
1	30	1	1	15
2	28	7	7	25
3	25	5	5	23
4	40	3	3	16
5	31	2	2	17
6	22	4	4	24
7	24	6	6	18
8	26	3	3	22
9	29	4	4	26
10	33	6	6	21
11	27	5	5	27
12	36	2	2	19
13	39	1	1	20
14	27	7	7	28
15	41	2	2	18
16	29	4	4	31
17	26	5	5	17
18	32	6	6	22
19	33	1	1	14
20	35	3	3	33

1. Автоматизувати запитання користувачеві про книги за допомогою циклу.

2. Зробити так, щоб користувач не міг залишити відповідь у вигляді порожнього рядка, скасувати відповідь або ввести назву книги довшу, ніж ***A*** символів. Якщо це відбувається – повертаємо користувача до питань знову. (До будь-якого рядка можна звернутися як *str.length* – і одержати його довжину).

3. За допомогою умов перевірити *personalBookDB.count*, і якщо він менший, ніж ***B*** – вивести повідомлення "Прочитано досить мало книг", якщо від ***C*** до ***D*** – "Ви класичний читач", а якщо більше – "Ви книгоман". А якщо не підійшло до жодного варіанта – "Сталася помилка".

4. Потренуватися та переписати цикл ще двома способами (попередні можна «закоментувати», або розмістити в окремих файлах).

5. Перевірити, щоб все працювало без помилок в консолі.

6. У кінці скрипту «закоментувати» дані про себе та виконану вами роботу.

Хід роботи:

1. Відкрийте попередню роботу (практ. роб. № 4).
2. Перейдіть до редагування JS-файла.
3. Замініть два послідовних запитання на цикл *for*.
4. Додайте перевірки на порожній рядок, скасування відповіді та довжину назви книги.
5. Додайте умовну конструкцію для перевірки кількості прочитаних книг.
6. Перепишіть цикл ще двома способами (наприклад, використовуючи цикли *while* та *do...while*):
7. Перевірте роботу скрипту та виправте помилки (якщо такі є).
8. Закоментуйте дані про себе та виконану вами роботу, наприклад:

```
// Практ. роб. № 5. Виконала студентка групи СТ-11 Васильєва Наталія.
```

9. Збережіть створені файли та, не змінюючи структури їх розташування, помістіть в архів із назвою PR-5-[група]-[прізвище транслітом].
10. Вивантажте архів у відповідну папку сховища, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Контрольні запитання:

1. Що таке умовні конструкції в JavaScript?
2. Наведіть приклад використання оператора *if...else*.
3. Що таке логічні оператори?
4. Наведіть приклад використання логічного оператора "І" (&&).
5. Що таке цикли в JavaScript?
6. Наведіть приклад використання циклу for.
7. Яка різниця між циклами *while* та *do...while*?

8. Як зупинити роботу циклу?

9. Як пропустити ітерацію циклу?

10. Що таке змінна циклу?

11. Як створити нескінченний цикл?

12. Що таке вкладені цикли?

13. Наведіть приклад використання оператора *switch*.

14. Як перевірити, чи належить значення до певного діапазону?

15. Що таке тест на рівність (===), і як він відрізняється від тесту на рівність (==)?

3.6 ПРАКТИЧНА РОБОТА № 6

Основи JavaScript: функції; callback-функції; методи і властивості рядків та чисел

Теоретичні відомості:

1. Функції

Функція в JavaScript – це набір інструкцій, які виконуються під час її виклику. Функції дозволяють структурувати код, роблячи його більш читабельним і легким для розуміння. Функції можуть приймати параметри та повертати значення.

Оголошення функції

Існує декілька способів оголошення функції в JavaScript:

1. Function Declaration (Оголошення функції):

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

2. Function Expression (Функціональний вираз):

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

3. Arrow Function (Стрілочна функція):

```
let sum = (a, b) => {
  return a + b;
};

// або скорочено
let sum = (a, b) => a + b;
```

Параметри та аргументи

Функція може приймати параметри, які є змінними, що передаються під час її виклику. Аргументи — це значення, які фактично передаються функції під час її виклику.

Приклад:

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

let result = sum(5, 10);
console.log(result); // 15
```

У цьому прикладі *a* та *b* — параметри функції *sum*, а 5 та 10 — аргументи, які передаються функції під час її виклику.

Повернення значення

Функція може повертати значення за допомогою оператора return. Значення, яке нею повертається, можна присвоїти змінній і використовувати далі в коді.

Приклад:

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

let result = sum(5, 10);
console.log(result); // 15
```

У прикладі функція *sum* повертає суму двох чисел, яка присвоюється змінній result.

Функції вищого порядку

Функції в JavaScript можуть приймати інші функції як параметри та повертати функції як значення. Такі функції називаються функціями вищого порядку.

Приклад:

```
function doWork(callback) {
  let result = "Результат виконання функції";
  callback(result);
}

function printMessage(message) {
  console.log(message);
}

doWork(printMessage);
```

У наведеному прикладі функція *doWork* приймає callback-функцію *printMessage* як параметр і викликає її всередині себе, передаючи їй результат своєї роботи.

Анонімні функції

Анонімна функція — це функція без назви. Анонімні функції часто використовують як callback-функції або в функціональних виразах.

Приклад:

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

console.log(sum(5, 10)); // 15
```

У цьому прикладі анонімна функція присвоюється змінній *sum* і використовується для обчислення суми двох чисел.

Область видимості

Змінні, оголошені всередині функції, мають локальну область видимості та недоступні ззовні функції. Змінні, оголошені ззовні функції, мають глобальну область видимості та доступні всередині функції.

Приклад:

```
let x = 10;

function foo() {
  let y = 20;
  console.log(x); // 10
}

foo();
console.log(y); // Uncaught ReferenceError: y is not defined
```

У прикладі змінна *x* має глобальну область видимості та доступна всередині функції foo, а змінна y має локальну область видимості та недоступна ззовні функції *foo*.

Замикання

Замикання — це функція, яка має доступ до змінних свого батьківського лексичного оточення навіть після завершення роботи батьківської функції.

Приклад:

```
function createCounter() {
  let count = 0;

  return function() {
    count++;
    console.log(count);
  };
}

let counter = createCounter();
counter(); // 1
counter(); // 2
counter(); // 3
```

У наведеному вище прикладі функція *createCounter* повертає анонімну функцію, яка має доступ до змінної *count* свого батьківського лексичного оточення. Таким чином, кожен виклик анонімної функції збільшує значення змінної *count* на *1*.

Рекурсія

Рекурсія — це виклик функцією самої себе. Рекурсію використовують для вирішення задач, які можна розбити на менші підзадачі.

Приклад:

```
function factorial(n) {
  if (n === 1) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}

console.log(factorial(5)); // 120
```

У цьому прикладі функція *factorial* обчислює факторіал числа *n* за допомогою рекурсії.

Оптимізація хвостової рекурсії

Оптимізація хвостової рекурсії — це техніка, яка дозволяє уникнути переповнення стеку під час виклику рекурсивної функції.

Приклад:

```
function factorial(n, acc = 1) {
  if (n === 1) {
    return acc;
  } else {
    return factorial(n - 1, n * acc);
  }
}

console.log(factorial(5)); // 120
```

Тут функція *factorial* оптимізована за допомогою хвостової рекурсії. Завдяки цьому, під час виклику функції з великим значенням n не відбувається переповнення стеку.

2. Callback-функції

Callback-функція — це функція, яка передається як аргумент іншій функції та викликається всередині неї. Callback-функції широко використовують у JavaScript для роботи з асинхронним кодом.

Приклад:

```
function printMessage(message) {
  console.log(message);
}

function doWork(callback) {
  let result = "Результат виконання функції";
  callback(result);
}

doWork(printMessage);
```

3. Методи та властивості рядків

Рядки в JavaScript мають декілька корисних методів і властивостей. Наприклад, метод length повертає довжину рядка, метод indexOf() шукає позицію підрядка всередині рядка, метод slice() вирізає частину рядка, метод toUpperCase() переводить рядок у верхній регістр тощо.

Приклад:

```
let str = "Hello, World!";
console.log(str.length); // 13
```

```
console.log(str.indexOf("World")); // 7
console.log(str.slice(0, 5)); // "Hello"
console.log(str.toUpperCase()); // "HELLO, WORLD!"
```

4. Методи та властивості чисел

Числа в JavaScript також мають декілька корисних методів і властивостей. Наприклад, метод toFixed() форматує число з вказаною кількістю десяткових знаків, метод toString() переводить число в рядок, властивість MAX_VALUE повертає максимальне значення числа в JavaScript тощо.

Приклад:

```
let num = 123.456;
console.log(num.toFixed(2)); // 123.46
console.log(num.toString()); // "123.456"
console.log(Number.MAX_VALUE); // 1.7976931348623157e+308
```

Мета роботи – освоїти навички роботи з функціями, callback-функціями, методами та властивостями рядків і чисел в JavaScript.

Завдання:

Використовуючи попередню роботу (практ. роб. № 5) відредагувати код таким чином:

1. Першу частину завдання (опитування щодо кількості книг, назви/оцінки та визначення рівня ..., класичний читач, ...) подати у вигляді трьох окремих функцій.

2. Створити функцію *showMyDB*, яка буде перевіряти властивість *privat*. Якщо вона (властивість) стоїть у позиції *false*, то виводить у консоль головний об'єкт.

3. Створити функцію *writeYourGenres*, у якій користувач буде 3 рази відповідати на питання "Ваш улюблений літературний жанр під номером ${номер по порядку}". Кожна відповідь записується в масив даних *genres*.

4. Перевірити, щоб все працювало без помилок в консолі.

5. У кінці скрипту «закоментувати» дані про себе та виконану вами роботу.

Хід роботи:

1. Відкрийте попередню роботу (практ. роб. №5).
2. Перейдіть до редагування JS-файла.
3. Виділіть першу частину завдання в окремі функції.
4. Створіть функцію *showMyDB*.
5. Створіть функцію *writeYourGenres*.
6. Перевірте роботу скрипту та виправте помилки (якщо такі є).
7. Закоментуйте дані про себе та виконану вами роботу, наприклад:

// Практична робота № 6. Виконав студент групи МВ-11 Петрук Ярослав.

8. Збережіть створені файли не змінюючи структури їх розташування та помістіть в архів із назвою PR-6-[група]-[прізвище транслітом].
9. Вивантажте архів у відповідну папку сховища, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Контрольні запитання:

1. Що таке функція в JavaScript?
2. Наведіть приклад використання функції з параметрами та поверненням значення.
3. Що таке callback-функція?
4. Наведіть приклад використання callback-функції.
5. Що таке методи рядків?
6. Наведіть приклад використання методу *indexOf()*.
7. Наведіть приклад використання методу *toFixed()*.
8. Як перевести число у рядок?
9. Як знайти довжину рядка?
10. Що таке властивості рядка?
11. Як перевести рядок у верхній регістр?
12. Що таке масиви?
13. Як додати елемент у масив?

3.7 ПРАКТИЧНА РОБОТА № 7

Основи JavaScript: методи об’єкта; методи (функції) для роботи з масивами; деструктуризація об’єктів

Теоретичні відомості:

1. Методи об'єкта

Методи об'єкта — це функції, які викликаються від імені об'єкта. Методи об'єкта мають доступ до властивостей і методів об'єкта за допомогою ключового слова *this*.

Створення методів об'єкта

Методи об'єкта можна створити двома способами:

1. Під час створення об'єкта:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010,
  showInfo: function() {
    console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
  }
};

car.showInfo(); // Brand: Toyota, Model: Corolla, Year: 2010
```

2. За допомогою прототипу об'єкта:

```
function Car(brand, model, year) {
  this.brand = brand;
  this.model = model;
  this.year = year;
}

Car.prototype.showInfo = function() {
  console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
};

let car = new Car("Toyota", "Corolla", 2010);
car.showInfo(); // Brand: Toyota, Model: Corolla, Year: 2010
```

Контекст виконання методів об'єкта

Контекст виконання методу об'єкта — це об'єкт, від імені якого викликається метод. Контекст виконання визначається автоматично під час виклику методу за допомогою крапки або квадратних дужок:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010,
  showInfo: function() {
    console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
  }
};

car.showInfo(); // Brand: Toyota, Model: Corolla, Year: 2010

let method = car.showInfo;
method(); // undefined, undefined, undefined (контекст виконання втрачено)

method.call(car); // Brand: Toyota, Model: Corolla, Year: 2010 (контекст виконання встановлено)
```

Зміна контексту виконання

Контекст виконання методу об'єкта можна змінити за допомогою методів *call(), apply() ma bind()*.

- *call()* — викликає функцію зі встановленим контекстом виконання та аргументами:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010
};

let car2 = {
  brand: "Honda",
  model: "Civic",
  year: 2015
};

function showInfo() {
  console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
}
```

```
showInfo.call(car); // Brand: Toyota, Model: Corolla, Year: 2010
showInfo.call(car2); // Brand: Honda, Model: Civic, Year: 2015
```

- *apply()* — викликає функцію зі встановленим контекстом виконання та аргументами, переданими у вигляді масиву:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010
};

let car2 = {
  brand: "Honda",
  model: "Civic",
  year: 2015
};

function showInfo(prefix) {
  console.log(`${prefix} Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
}

showInfo.apply(car, ["Car: "]); // Car: Brand: Toyota, Model: Corolla, Year: 2010
showInfo.apply(car2, ["Car: "]); // Car: Brand: Honda, Model: Civic, Year: 2015
```

- *bind()* — створює нову функцію зі встановленим контекстом виконання та аргументами:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010
};

let car2 = {
  brand: "Honda",
  model: "Civic",
  year: 2015
};

function showInfo() {
  console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
}

let showCarInfo = showInfo.bind(car);
```

```
let showCar2Info = showInfo.bind(car2);

showCarInfo(); // Brand: Toyota, Model: Corolla, Year: 2010
showCar2Info(); // Brand: Honda, Model: Civic, Year: 2015
```

Передача аргументів у методи об'єкта

Аргументи в методи об'єкта можна передавати так само, як і у звичайні функції:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010,
  showInfo: function(prefix) {
    console.log(`${prefix} Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
  }
};

car.showInfo("Car: "); // Car: Brand: Toyota, Model: Corolla, Year: 2010
```

Область видимості методів об'єкта

Змінні, оголошені всередині методів об'єкта, мають локальну область видимості та недоступні ззовні методу. Змінні, оголошені ззовні методу, мають глобальну область видимості та доступні всередині методу.

Приклад:

```
let x = 10;

let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010,
  showInfo: function() {
    let y = 20;
    console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}, X: ${x}, Y: ${y}`);
  }
};

car.showInfo(); // Brand: Toyota, Model: Corolla, Year: 2010, X: 10, Y: 20
console.log(y); // Uncaught ReferenceError: y is not defined
```

У цьому прикладі змінна x має глобальну область видимості та доступна всередині методу *showInfo*, а змінна y має локальну область видимості та недоступна ззовні методу *showInfo*.

Використання стрілочних функцій як методів об'єкта

Стрілочні функції не мають власного контексту виконання *this*, тому вони не підходять для використання як методи об'єкта:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010,
  showInfo: () => {
    console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
  }
};

car.showInfo(); // undefined, undefined, undefined (контекст виконання втрачено)
```

Замість цього, можна використовувати звичайну функцію або метод *bind()*:

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  year: 2010,
  showInfo: function() {
    console.log(`Brand: ${this.brand}, Model: ${this.model}, Year: ${this.year}`);
  }
};

car.showInfo(); // Brand: Toyota, Model: Corolla, Year: 2010

let showCarInfo = car.showInfo.bind(car);
showCarInfo(); // Brand: Toyota, Model: Corolla, Year: 2010
```

Отже:

Методи об'єкта — це функції, які викликаються від імені об'єкта. Методи об'єкта мають доступ до властивостей і методів об'єкта за допомогою ключового слова *this*. Контекст виконання методу об'єкта визначається автоматично під час виклику методу за допомогою крапки або квадратних дужок, але його можна змінити за допомогою методів *call(), apply() та bind()*. Змінні, оголошені всередині методів об'єкта, мають локальну область видимості та недоступні ззовні

методу. Стрілочні функції не мають власного контексту виконання *this*, тому вони не підходять для використання як методів об'єкта.

2. Методи (функції) для роботи з масивами

JavaScript надає багато корисних методів для роботи з масивами. Деякі з них:

- *push()* — додає елемент у кінець масиву;
- *pop()* — видаляє останній елемент з масиву;
- *shift()* — видаляє перший елемент з масиву;
- *unshift()* — додає елемент на початок масиву;
- *indexOf()* — шукає позицію елемента в масиві;
- *slice()* — вирізає частину масиву;
- *splice()* — видаляє або замінює елементи в масиві;
- *sort()* — сортує елементи масиву;
- *forEach()* — виконує функцію для кожного елемента масиву;
- *map()* — створює новий масив, застосовуючи функцію до кожного елемента поточного масиву;
- *filter()* — створює новий масив, який містить тільки ті елементи, для яких функція повертає true;
- *reduce()* — зводить масив до одного значення, використовуючи функцію.

Приклад:

```
let arr = [1, 2, 3, 4, 5];

arr.push(6);
console.log(arr); // [1, 2, 3, 4, 5, 6]

arr.pop();
console.log(arr); // [1, 2, 3, 4, 5]

arr.shift();
console.log(arr); // [2, 3, 4, 5]

arr.unshift(1);
console.log(arr); // [1, 2, 3, 4, 5]
```

```
console.log(arr.indexOf(3)); // 2

console.log(arr.slice(1, 3)); // [2, 3]

arr.splice(2, 1);
console.log(arr); // [1, 2, 4, 5]

arr.sort();
console.log(arr); // [1, 2, 4, 5]

arr.forEach(function(item, index, array) {
  console.log(`Index: ${index}, Item: ${item}, Array: ${array}`);
});

let newArr = arr.map(function(item) {
  return item * 2;
});
console.log(newArr); // [2, 4, 8, 10]

let filteredArr = arr.filter(function(item) {
  return item > 2;
});
console.log(filteredArr); // [4, 5]

let sum = arr.reduce(function(accumulator, item) {
  return accumulator + item;
}, 0);
console.log(sum); // 12
```

3. Деструктуризація об'єктів

Деструктуризація об'єктів — це синтаксис JavaScript, який дозволяє витягувати значення властивостей об'єкта в змінні.

Приклад:

```
let user = {
  name: "John",
  age: 30,
  email: "john@example.com"
};

let { name, age, email } = user;
console.log(name); // John
console.log(age); // 30
console.log(email); // john@example.com
```

Мета роботи – освоїти навички роботи з методами об'єкта, методами (функціями) для роботи з масивами та деструктуризацією об'єктів у JavaScript.

Завдання:

1. З огляду на попередню роботу (практична робота № 6) у нас вже є дієвий додаток, який складається з окремих функцій. Необхідно переписати його так, щоб усі функції стали методами об'єкта *personalBookDB* (Таке трапляється в реальних продуктах при зміні технологій або підходу до архітектури програми).
2. Створити метод *toggleVisibleMyDB*, який під час виклику перевірятиме властивість *private*. Якщо вона *false* – він перемикає її в *true*, якщо *true* – перемикає в *false*. Протестувати разом із *showMyDB*.
3. У методі *writeYourGenres* заборонити користувачеві натиснути кнопку "скасувати" або залишати порожній рядок. Якщо він це зробив – повернути його до цього питання. Після того, як усі жанри введено – за допомогою методу *forEach* вивести в консоль повідомлення в такому вигляді: "**Улюблений жанр** #(номер по порядку, починаючи з 1) **– це** (назва з масиву)".
4. Перевірити, щоб усе працювало без помилок у консолі.

Хід роботи:

1. Відкрийте попередню роботу (практ. роб. № 6).
2. Перейдіть до редагування JS-файла.
3. Перепишіть функції як методи об'єкта *personalBookDB*.
4. Створіть метод *toggleVisibleMyDB*, який перевіряє властивість *private* та перемикає її значення.
5. Здійсніть модифікацію метода *writeYourGenres*, додаючи перевірки на скасування та порожній рядок. Використовуйте метод *forEach* для виведення повідомлень про улюблені літературні жанри.
6. Перевірте роботу скрипту та виправіть помилки (якщо такі є).
7. Закоментуйте дані про себе та виконану вами роботу, наприклад:

```
// Практична робота № 7. Виконав студент групи СТ-12 Глущенко В'ячеслав.
```

8. Збережіть створені файли та, не змінюючи структуру їх розташування, помістіть в архів із назвою PR-7-[група]-[прізвище транслітом].
9. Вивантажте архів у відповідну папку сховища, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Контрольні запитання:

1. Що таке методи об'єкта?
2. Наведіть приклад використання методу об'єкта.
3. Що таке деструктуризація об'єктів?
4. Наведіть приклад використання деструктуризації об'єктів.
5. Що таке масиви?
6. Наведіть приклад використання методу *push()*.
7. Що таке метод *forEach()*?
8. Наведіть приклад використання методу *map()*.
9. Що таке метод *filter()*?
10. Наведіть приклад використання методу *reduce()*.
11. Що таке *this*?
12. Як отримати доступ до властивостей об'єкта?

3.8 ПРАКТИЧНА РОБОТА № 8

Основи JavaScript: отримання елементів з вебсторінки; дії з елементами на вебсторінці

Теоретичні відомості:

Отримання елементів з вебсторінки

JavaScript надає багато методів і властивостей для отримання елементів з вебсторінки. Ці методи та властивості дозволяють знайти елементи за різними критеріями, такими як ідентифікатор, клас, назва тегу, CSS-селектор тощо. Отримані елементи можна використовувати для виконання різних дій з ними, зокрема таких як зміна вмісту, стилів, атрибутів та інших властивостей.

Для отримання елементів з вебсторінки в JavaScript використовують спеціальні методи та властивості об'єкта *document*, які дозволяють знайти елементи за різними критеріями, як-от: ідентифікатор (*id*), клас (*class*), назва тегу (*tagName*), CSS-селектор та інші.

1) document.getElementById()

Метод *document.getElementById()* повертає елемент з указаним ідентифікатором (*id*). Ідентифікатор має бути унікальним на сторінці, тому цей метод завжди повертає єдиний елемент.

Приклад:

```
<div id="myDiv">Мій дів</div>

<script>
let div = document.getElementById("myDiv");
console.log(div); // <div id="myDiv">Мій дів</div>
</script>
```

2) document.getElementsByClassName()

Метод *document.getElementsByClassName()* повертає колекцію елементів із вказаним класом (*class*). Клас може бути присвоєний декільком елементам на сторінці, тому цей метод може повернути декілька елементів.

Приклад:

```
<div class="myClass">Мій дів 1</div>
<div class="myClass">Мій дів 2</div>

<script>
let divs = document.getElementsByClassName("myClass");
console.log(divs); // HTMLCollection(2) [div.myClass, div.myClass]
console.log(divs[0]); // <div class="myClass">Мій дів 1</div>
console.log(divs[1]); // <div class="myClass">Мій дів 2</div>
</script>
```

3) document.getElementsByTagName()

Метод *document.getElementsByTagName()* повертає колекцію елементів з указаною назвою тегу (*tagName*). Назва тегу може бути присвоєна декільком елементам на сторінці, тому цей метод може повернути декілька елементів.

Приклад:

```
<div>Мій дів 1</div>
<div>Мій дів 2</div>

<script>
let divs = document.getElementsByTagName("div");
console.log(divs); // HTMLCollection(2) [div, div]
console.log(divs[0]); // <div>Мій дів 1</div>
console.log(divs[1]); // <div>Мій дів 2</div>
</script>
```

4. document.querySelector()

Метод *document.querySelector()* повертає перший елемент, який відповідає вказаному CSS-селектору. CSS-селектор може містити будь-які критерії добору елементів – ідентифікатор, клас, назва тегу, атрибут тощо.

Приклад:

```
<div id="myDiv" class="myClass">Мій дів</div>

<script>
let div = document.querySelector("#myDiv");
console.log(div); // <div id="myDiv" class="myClass">Мій дів</div>

div = document.querySelector(".myClass");
console.log(div); // <div id="myDiv" class="myClass">Мій дів</div>
```

```
div = document.querySelector("div");
console.log(div); // <div id="myDiv" class="myClass">Мій дів</div>
</script>
```

5. document.querySelectorAll()

Метод *document.querySelectorAll()* повертає колекцію елементів, які відповідають указаному CSS-селектору. CSS-селектор може містити будь-які критерії добору елементів, зокрема ідентифікатор, клас, назва тегу, атрибут тощо.

Приклад:

```
<div class="myClass">Мій дів 1</div>
<div class="myClass">Мій дів 2</div>

<script>
let divs = document.querySelectorAll(".myClass");
console.log(divs); // NodeList(2) [div.myClass, div.myClass]
console.log(divs[0]); // <div class="myClass">Мій дів 1</div>
console.log(divs[1]); // <div class="myClass">Мій дів 2</div>
</script>
```

6. document.forms

Властивість *document.forms* повертає колекцію всіх форм на вебсторінці. Кожна форма має властивість *elements*, яка містить колекцію всіх елементів форми.

Приклад:

```
<form name="myForm">
  <input type="text" name="myInput">
  <button type="submit">Відправити</button>
</form>

<script>
let form = document.forms.myForm;
console.log(form); // <form name="myForm">...</form>

let input = form.elements.myInput;
console.log(input); // <input type="text" name="myInput">
</script>
```

7. document.images

Властивість *document.images* повертає колекцію всіх зображень на вебсторінці.

Приклад:

```
<img src="image.jpg" alt="Зображення">

<script>
let images = document.images;
console.log(images); // HTMLCollection(1) [img]
console.log(images[0]); // <img src="image.jpg" alt="Зображення">
</script>
```

8. document.links

Властивість *document.links* повертає колекцію всіх посилань на вебсторінці.

Приклад:

```
<a href="https://example.com">Посилання</a>

<script>
let links = document.links;
console.log(links); // HTMLCollection(1) [a]
console.log(links[0]); // <a href="https://example.com">Посилання</a>
</script>
```

9. document.body

Властивість *document.body* повертає елемент <*body*> вебсторінки.

Приклад:

```
<body>
  <div>Мій дів</div>
</body>

<script>
let body = document.body;
console.log(body); // <body>...</body>
console.log(body.children[0]); // <div>Мій дів</div>
</script>
```

10. document.documentElement

Властивість *document.documentElement* повертає елемент <*html*> вебсторінки.

Приклад:

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

```
</html>

<script>
let html = document.documentElement;
console.log(html); // <html>...</html>
console.log(html.children[1]); // <body>...</body>
</script>
```

Дії з елементами на вебсторінці

У JavaScript можна виконувати різні дії з елементами на вебсторінці: створювати, видаляти, змінювати їх вміст і стилі, додавати й видаляти класи, атрибути та властивості. Ці дії дозволяють динамічно змінювати контент і дизайн вебсторінки залежно від подій та умов. Для виконання цих дій використовуються спеціальні методи та властивості об'єкта *document*.

1. Створення елементів

Для створення нового елемента на вебсторінці використовують метод *createElement()* об'єкта *document*. Цей метод приймає один параметр — назву тегу нового елемента.

Приклад:

```
let div = document.createElement("div");
console.log(div); // <div></div>
```

2. Додавання елементів

Для додавання нового елемента на вебсторінку необхідно викликати метод *appendChild()* у батьківського елемента. Цей метод приймає один параметр — елемент, який необхідно додати.

Приклад:

```
let body = document.body;
let div = document.createElement("div");
div.textContent = "Мій дів";
body.appendChild(div);
```

3. Видалення елементів

Для видалення елемента з вебсторінки необхідно викликати метод *remove()* у цього елемента.

Приклад:

```
let div = document.querySelector("div");
div.remove();
```

4. Зміна вмісту елементів

Для зміни вмісту елемента використовують властивості *textContent*, *innerHTML* та *innerText*.

- *textContent* — змінює текстовий вміст елемента, включно з пробілами та переносами рядків.
- *innerHTML* — змінює вміст елемента, включаючи HTML-теги.
- *innerText* — змінює текстовий вміст елемента, але з урахуванням стилів.

Приклад:

```
let div = document.querySelector("div");
div.textContent = "Новий текст";
div.innerHTML = "<strong>Новий текст</strong>";
div.innerText = "Новий\nтекст";
```

5. Зміна стилів елементів

Для зміни стилів елемента використовують властивість *style*. Ця властивість дозволяє змінювати будь-які CSS-стилі елемента.

Приклад:

```
let div = document.querySelector("div");
div.style.color = "red";
div.style.fontSize = "24px";
div.style.backgroundColor = "gray";
```

6. Додавання та видалення класів

Для додавання та видалення класів елемента використовують методи *classList.add()* і *classList.remove()*.

- *classList.add()* — додає клас елементові.
- *classList.remove()* — видаляє клас елемента.

Приклад:

```
let div = document.querySelector("div");
div.classList.add("myClass");
div.classList.remove("myClass");
```

7. Перевірка наявності класу

Для перевірки наявності класу елемента використовують метод *classList.contains()*.

- *classList.contains()* — перевіряє, чи містить елемент указаний клас.

Приклад:

```
let div = document.querySelector("div");
console.log(div.classList.contains("myClass")); // false або true
```

8. Додавання та видалення атрибутів

Для додавання та видалення атрибутів елемента використовують методи *setAttribute()* та *removeAttribute()*.

- *setAttribute()* — додає атрибут елементові.
- *removeAttribute()* — видаляє атрибут елемента.

Приклад:

```
let a = document.querySelector("a");
a.setAttribute("href", "https://example.com");
a.removeAttribute("href");
```

9. Отримання значення атрибуту

Для отримання значення атрибута елемента використовують властивості *getAttribute()* та *hasAttribute()*.

- *getAttribute()* — отримує значення атрибута елемента.
- *hasAttribute()* — перевіряє, чи містить елемент зазначений атрибут.

Приклад:

```
let a = document.querySelector("a");
console.log(a.getAttribute("href")); // https://example.com або null
console.log(a.hasAttribute("href")); // true або false
```

10. Зміна властивостей елементів

Для зміни властивостей елементів використовують відповідні властивості об'єкта.

Приклад:

```
let input = document.querySelector("input");
input.type = "text";
input.value = "Текст";
```

```
input.checked = true;
input.disabled = true;
```

Мета роботи – освоїти навички отримання елементів з вебсторінки та виконання різних дій з ними за допомогою JavaScript.

Завдання:

1. Видобути архів PR-8.rar, наданий викладачем.
2. Відкрити файл ...\project\js\script.js для редагування і ...\project\index.html для перегляду вмісту та коду.
3. Видалити всі рекламні блоки на сторінці (права частина сайту).
4. Змінити жанр фільму, замінити "комедія" на "драма".
5. Змінити задній фон постера з фільмом на зображення "bg.jpg", яке розташоване в папці img.
6. На основі даних із наданого JS файлу сформувати на сторінці список фільмів. Сортувати за алфавітом.
7. Додати нумерацію виведених фільмів.

Хід роботи:

1. Видобуваємо архів PR-8.rar в окрему папку.
2. Відкриваємо файл ...\project\js\script.js для редагування і ...\project\index.html для перегляду вмісту та коду.
3. Видаляємо всі рекламні блоки на сторінці (права частина сайту) за допомогою методу *remove()*.
4. Змінюємо жанр фильму, замінюючи "комедія" на "драма" за допомогою властивості *textContent*.
5. Змінюємо задній фон постера з фільмом на зображення "bg.jpg", яке розташоване в папці img, за допомогою властивості *style*.
6. Список фільмів на сторінці формуємо на основі даних із наданого JS файлу. Сортуємо за алфавітом за допомогою методу *sort()*.

7. Додаємо нумерацію виведених фільмів за допомогою змінної-лічильника.
8. Перевірте роботу скрипту та виправіть помилки (якщо такі є).
9. Закоментуйте дані про себе та виконану вами роботу, наприклад:

// Практична робота № 8. Виконав студент групи МВ-11 Мельник Андрій.

10. Збережіть файли та, не змінюючи структуру їх розташування, помістіть в архів із назвою PR-8-[група]-[прізвище транслітом].
11. Вивантажте архів у відповідну папку сховища, або перешліть чи передайте на електронному носії викладачеві для подальшої перевірки роботи.

Контрольні запитання:

1. Що таке методи *getElementById(), getElementsByClassName(), getElementsByTagName(), querySelector(), querySelectorAll()*?
2. Наведіть приклад використання методу *createElement()*.
3. Що таке властивість *innerHTML*?
4. Що таке властивість *style*?
5. Як додати клас елементові?
6. Що таке методи *setAttribute()*, *getAttribute()*?
7. Як видалити елемент з вебсторінки?
8. Що таке метод *sort()*?
9. Як додати нумерацію виведених елементів?
10. Що таке CSS-селектори?
11. Як отримати доступ до елементів, які відповідають вказаному CSS-селектору?
12. Що таке колекція елементів?
13. Як перебрати колекцію елементів?
14. Що таке властивість *textContent*?
15. Як змінити текстовий вміст елемента?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова література

1. Flanagan D. JavaScript the definitive guide / David Flanagan.. – 1245 p. – (7). P. 38–39.
2. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript.
3. Pros and Cons of JavaScript – Weigh them and Choose wisely! [Електронний ресурс] – Режим доступу до ресурсу: https://data-flair.training/blogs/advantages-disadvantages-javascript/.
4. Dickson B. A Brief History of JavaScript [Електронний ресурс] / Boateng Dickson. – 2022. – Режим доступу до ресурсу: https://dev.to/dboatengx/history-of-javascript-how-it-all-began-92a.
5. OpenJS Foundation. About Node.js [Електронний ресурс] / OpenJS Foundation – Режим доступу до ресурсу: https://nodejs.org/en/about.
6. Introduction [Електронний ресурс] // ECMAScript – Режим доступу до ресурсу: https://262.ecma-international.org/13.0/#sec-intro.
7. Deed I. Pros and Cons of JavaScript Development. 2023 | Hire Flutter, Mobile & JS Software Devs | Pangea.ai. https://www.pangea.ai/dev-javascript-resources/best-practices/
8. JavaScript | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
9. Classes – JavaScript | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes.
10. Property accessors — JavaScript | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_accessors.

Допоміжна література

11. String.prototype.replace() – JavaScript | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace (дата звернення 05.05.2023).

12. Introduction to Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/en/learn/.

13. An introduction to the NPM package manager [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/en/learn/an-introduction-to-the-npm-package-manager/.

14. GeeksforGeeks. (2023). Advantages and Disadvantages of JavaScript. GeeksforGeeks. Режим доступу до ресурсу: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-javascript/

15. Świstak, T. (2023, January 2). What Is TypeScript? Pros and Cons of TypeScript vs. JavaScript. STXNEXT. Режим доступу до ресурсу: https://www.stxnext.com/blog/typescript-pros-cons-javascript/

16. Senecki, A. (2023). TypeScript vs JavaScript comparison – pros, cons, trends. The Software House. Режим доступу до ресурсу: https://tsh.io/blog/typescript-vs-javascript-comparison/

17. Sharma, A. (2023). What Is Express JS In Node JS? Simplilearn.com. Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js

18. GeeksforGeeks. (2023). Express.js. GeeksforGeeks. Режим доступу до ресурсу: https://www.geeksforgeeks.org/express-js/

19. Rashevskaya A. Node.js Architecture From A to Z [Електронний ресурс] / Anastasia Rashevskaya. – 2021. – Режим доступу до ресурсу: https://litslink.com/blog/node-js-architecture-from-a-to-z.

ДОДАТОК А

ПРИКЛАД ТИТУЛЬНОГО АРКУШУ ДО ПРАКТИЧНИХ ЗАВДАНЬ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий видавничо-поліграфічний інститут

Кафедра репрографії

Практична робота
з дисципліни
«Конструювання прототипів і шаблонів вебсторінок»
на тему:

«_____»

Виконав

Студент групи ___: ПІБ

Перевірив

к.т.н., доц.: Тріщук Руслан Любомирович

Київ-20__