Міністерство освіти та науки України
Чернівецький національний університет
імені Юрія Федьковича

Вебтехнології та вебпрограмування

**Методичні вказівки та завдання
до лабораторних робіт**

Чернівці
Чернівецький національний університет
2023

УДК 004.774(076.5)
В260

Рекомендовано до друку Вченою радою факультету
математики та інформатики Чернівецького національного університету
імені Юрія Федьковича
(протокол № 6 від « 25 » січня 2023 року)

Рецензенти:
Піддубна Лариса Андріївна, доцент кафедри математичного моделювання;
Матвій Олександр Васильович, доцент кафедри математичного моделювання.

В260 Готинчан Т.І. Вебтехнології та вебпрограмування: методичні вказівки та завдання до лабораторних робіт / Укл: Т.І. Готинчан, – Чернівці : Чернівецький нац. ун-т, 2023. – 56 с.

Видання містить методичні вказівки та завдання до лабораторних робіт, які охоплюють основні принципи вступу до веброзробки за допомогою HTML, CSS, JS та його бібліотек.

Для студентів спеціальностей «Комп'ютерні науки» та «Системний аналіз»

УДК 004.774(076.5)

Зміст

Вступ

Реалії сьогодення людства вимагають створення різноманітних вебпродуктів, які допомагають людині у різних сферах її діяльності. Веброзробка постійно змушує розробників постійно вивчати щось нове, щоб бути кращим і затребуваним. Її величезний плюс – це "низький" поріг входу у професію. Так, вивчати потрібно буде багато всього, і робити це потрібно постійно. Але для того, щоб «почати», потрібно вивчити ті три стовпи веб-розробки:

- HTML;
- CSS;
- JavaScript.

Перераховані вище технології – лише необхідна база. HTML – стандартизована мова розмітки сторінок в інтернеті, CSS – каскадна таблиця стилів, що відповідає за зовнішній вигляд, мова програмування JavaScript відповідає за реагування елементів на дії користувача.

Найкраще, якщо освоєння основ веброзробки відбувається в умовах реального проекту – тоді відбувається освоєння й супутніх інструментів: графічних редакторів та редакторів коду, інструментів розробника у браузері тощо. У попередньому курсі «Основи інтернет-технологій» студенти ознайомились із сучасними підходами HTML і CSS. Тепер вони опановуватимуть JavaScript.

Опанування сучасних підходів мови JavaScript у веброзробці на стороні клієнта допомагають студентам спеціальностей «Комп'ютерні науки» та «Системний аналіз» галузі «Інформаційні технології» у майбутньому професійно створювати вебпродукти різного рівня складності.

Методичні вказівки містять завдання до лабораторних робіт з курсу «Вебтехнології та вебпрограмування», а також теоретичні відомості.

Перша лабораторна робота є вступною із вивчення JavaScript. Розв'язуючи алгоритмічні задачі, студенти опановують роботу із змінними, вбудованими об'єктами та операторами.

Починаючи з другої лабораторної роботи, кожна з них є основою для наступної. Під час їх виконання студенти опановують навики роботи з DOM, BOM, функціонального програмування, роботи з класами, асинхронного JavaScript та використання JavaScript бібліотек. Причому усі коди підпорядковуються одній темі проєкту. Під час роботи з DOM студенти повторюють раніше вивчені теми, пов’язані з HTML і CSS.

Таким чином, студенти виконують індивідуальні проєкти за вибраною тематикою. Наприкінці курсу вони розміщують свої проєкти на одному з безплатних хостингів.

Мета і завдання навчальної дисципліни

Мета навчальної дисципліни полягає в опануванні майбутніми фахівцями теоретичних знань і практичних навичок, необхідних для вирішення питань, пов'язаних із верстkою вебсторінок у глобальній мережі інтернет з використанням сучасних технологій, а саме навчити студентів основним аспектам мови сценарію JS, роботи з об'єктною моделлю документа (DOM) та об'єктною моделлю браузера (BOM); використовувати сучасні бібліотеки JS роботи з датами, графікою, анімацією; бібліотеки JQuery та JQuery UI.

Дисципліна формує такі компетентності [1, 2]:

- за ОП «Комп'ютерні науки»:

Загальні компетентності:
ЗК1. Здатність до абстрактного мислення, аналізу та синтезу.
ЗК2. Здатність застосовувати знання у практичних ситуаціях.
ЗК7. Здатність до пошуку, оброблення та аналізу інформації з різних джерел.
ЗК8. Здатність генерувати нові ідеї (креативність).

Фахові компетентності спеціальності:
ФК3. Здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проектування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв'язності та нерозв'язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем.
ФК10. Здатність застосовувати методології, технології та інструментальні засоби для управління процесами життєвого циклу інформаційних і програмних систем, продуктів і сервісів інформаційних технологій відповідно до вимог замовника.

Наведені результати навчання за відповідною дисципліною співвідносяться із ***такими програмними результатами навчання***:

ПРН1. Застосовувати знання основних форм і законів абстрактно-логічного мислення, основ методології наукового пізнання, форм і методів вилучення, аналізу, обробки та синтезу інформації в предметній області комп'ютерних наук.
ПРН10. Використовувати інструментальні засоби розробки клієнт-серверних застосувань.

- ОП «Системний аналіз»:

Загальні компетентності:
ЗК1. Здатність до абстрактного мислення, аналізу та синтезу.
ЗК2. Здатність застосовувати знання у практичних ситуаціях.
ЗК7. Здатність до пошуку, оброблення та аналізу інформації з різних джерел.
ЗК11. Здатність генерувати нові ідеї (креативність).
ЗК14. Здатність оцінювати та забезпечувати якість виконуваних робіт

Фахові компетентності спеціальності:
ФК8. Здатність організовувати роботу з аналізу та проектування складних систем, створення відповідних інформаційних технологій та програмного забезпечення.
ФК11. Здатність системно аналізувати свою професійну і соціальну діяльність, оцінювати накопичений досвід.

Наведені результати навчання за відповідною дисципліною співвідносяться із такими ***програмними результатами навчання*:**
ПР8. Володіти сучасними методами розробки програм і програмних комплексів та прийняття оптимальних рішень щодо складу програмного забезпечення, алгоритмів процедур і операцій.
ПР13. Проектувати, реалізовувати, тестувати, впроваджувати, супроводжувати, експлуатувати програмні засоби роботи з даними і знаннями в комп’ютерних системах і мережах.

Завдання та вказівки до лабораторних робіт

Лабораторна робота № 1.

Основи використання JS

Короткі теоретичні відомості

JavaScript – скриптова мова загального призначення, що відповідає специфікації ECMAScript. Її використовують як на боці клієнтської частини, так і на боці сервера.

JavaScript – це заснована на прототипі, багатопарадигмальна, однопотокова, динамічна мова, яка підтримує об'єктно-орієнтований, імперативний і декларативний (наприклад, функціональне програмування) стилі. Динамічні можливості JavaScript включають конструкцію об'єктів під час виконання, списки змінних параметрів, змінні функцій, динамічне створення сценаріїв, самоаналіз об'єктів та відновлення вихідного коду [3].

Підключення скриптів

Можливі такі способи підкючення скриптів:

- інлайнове;
- вбудоване;
- зовнішнє.

Інлайнове

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title> Назва </title>
</head>
<body>
```

```
  <div onclick= “виклик JavaScript-код”>… </div>
  <script>
    // JavaScript-код
  </script>
</body>
</html>
```

Зауваження 1. При цьому типі підключення HTML сторінка починає відображатися у браузері, але як тільки на шляху завантаження сторінки попадається тег <script> починається виконання JavaScript коду, а HTML сторінка продовжить своє завантаження після повного виконання JavaScript коду**.**

Вбудоване

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Назва</title>
  <script>
    // JavaScript-код
  </script>
</head>
<body>
...
</body>
</html>
```

Зауваження 2. При цьому типі підключення HTML сторінка починає відображатися у браузері лише після виконання JavaScript коду**.**

Зовнішнє

```
<!DOCTYPE html>
<html>
```

```
<head>
  <meta charset="utf-8">
  <title>Назва</title>
  <script src="myscript1.js"></script>
</head>
<body>
...
  <script src="myscript2.js"></script>
</body>
</html>
```

Зауваження 3. Є хороша практика під'єднувати скрипти унизу сторінки. Тоді вся сторінка і стилі завантажаться повністю без затримки.

Крім розміщення файлів в кінці HTML сторінки, існують способи змінити стандартну поведінку завантаження, зробивши її асинхронною за допомогою атрибутів defer або async.

defer

```
<script src="[path]/script.js" defer></script>
```

Завантаження скриптів починається разом із завантаженням сторінки, проте виконання відбувається після повного її завантаження. При зазначенні декількох зовнішніх JavaScript файлів з атрибутом defer, файли завантажуються у тій послідовності, в якій вони задані.

```
<script src="first.js" defer></script>
<script src="second.js" defer></script>
```

async

```
<script src="[path]/script.js" async></script>
```

Завантаження скриптів починається разом із завантаженням сторінки, виконання відбувається після завантаження скрипта, можливе призупинення завантаження сторінки.

При зазначенні декількох зовнішніх JavaScript файлів з атрибутом async, порядок їх виконання залежить від швидкості завантаження кожного, при цьому порядок їх розміщення не важливий. Тобто, який файл перший завантажиться, той перший і виконається.

```
<script src="first.js" async></script>
<script src="second.js" async></script>
```

Змінні

Ключові слова для оголошення змінних: **var , let, const.**

Особливості var:

- спосіб оголошення існував завжди (цей спосіб існує з першої версії JavaScript);
- піднімаються (функціонує механізм hoisting);
- функціональна область видимості;
- можна оголосити змінну з таким же ім'ям (перевизначити) (але не в режимі "use strict").

Особливості let:

- спосіб оголошення було додано в специфікацію ES6;
- не піднімаються (відсутнє поняття hoisting);
- блокова область видимості (всередині фігурних дужок);
- не можна оголосити змінну з таким же ім'ям (перевизначити).

Особливості const:

- не піднімаються (відсутнє поняття hoisting);
- не можна оголосити змінну з таким же ім'ям;
- не можна змінити значення змінної;
- не можна оголосити змінну без значення (ініціалізацію).

Рекомендується використовувати для оголошення та ініціалізації змінних ключове слово **let**.

Наприклад**,**

```
let firstName = ”Mykola”,
    lastName = ‘Prizvyschenko’,
    age = 20;

let let isSelected = true;
const  g = 9.81;
```

У JavaScript типи даних можна розділити на дві категорії: **прості** (примітивні) типи і **складові** (посилальні або об'єктні).

До категорії **простих** типів відносяться:

- string - текстові рядки (зазвичай їх називають просто - рядки);
- number - числа;
- boolean - логічні (булеві) значення.
- null;
- undefined

До **складових** типів даних відносяться object - об'єкти.

У специфікаціїї ES2015 введений тип даних, який служить для створення унікальних ідентифікаторів – Symbol. У деяких мовах програмування символи також називаються атомами.

Оголошувати змінні типу string, number, boolean слід літерально. При цьому ініціюються вбудовані об’єкти String Number Boolean, які надають доступ до властивостей та методів [3-6].

Якщо значення змінної типу string дуже довгі рядки, то краще використовувати символьний рядок. Відображення – як вводиться. Зручно також використовувати параметри.

Наприклад,

1)
```
let text = ` It's some long
    text
    about string `;
```

2) let name = ”Taras”,
lastName = “Shevcshenko”;

let divOut = document.querySelector(“#div-out”);

divOut.innerHTML = `<p class=”text”> ${ name} ${ lastName} – видатний український поет, письменник і художник .</p>`

Змінна типу object може складатися з рядків, чисел, логічних значень і т.д. Зазвичай це якась колекція або набір даних, які містять у собі **ключ: значення**.

Оголошуються об'єкти за допомогою фігурних дужок {...}, наприклад:

```
let obj = {
  id: 1,
  name: "Mykola",
  active: true
}
```

У JavaScript немає жорсткої типізації, це мова з динамічною типізацією даних, що призводить до того, що в певних випадках, дані одного типу можуть бути неявно перетворені в інший тип автоматично.

Є три типи перетворень:

• строкове перетворення;

• чисельне перетворення;

• логічне перетворення.

Оператор typeof служить для визначення типу. Результатом роботи typeof є рядок, що містить тип.

Оператори

- **Присвоєння** =

let calculated = (12 + 8) * 2; // 40

- **Бінарні операції** +, *, -, /, %, **

Остача від ділення (%) повертає цілий залишок від ділення лівого операнда на правий. Результат ділення із залишком матиме той же знак, що і перший операнд

Оператор ** (піднесення до степеня) має два операнда. Перший операнд є підставою ступеня, другий операнд - показником ступеня, в результаті оператор повертає підставу, зведена в зазначену ступінь:

```
let m = 2 ** 4; // 16
let n = 10;  n *= 5;  // 50
```

- **Унарний оператор +**, тобто застосовується до одного операнду, перетворює рядки до числа.

```
let one = + "50"; // 50
let two = 12 + "70"; // "1270"
let three = 12 +  + "70"; // 82
```

- **Логічні оператори**

оператор || (Логічне АБО)
Якщо хоча б один з аргументів виразу «обчислюється» як true, то весь вираз обчислюється як true.
Оператор && (логічне І)
Якщо хоча б один з аргументів виразу «обчислюється» як false, то весь вираз обчислюється як false.
Оператор ! (Логічне НЕ)
Оператор ! перетворює (якщо необхідно) значення свого операнда до логічного типу, потім інвертує це значення в протилежне і повертає отримане в результаті інвертування логічне значення.
Подвійний оператор НЕ (!!)

Оператор !! перетворює операнд до логічного типу, шляхом подвійного послідовного інвертора.

- **Умовні конструкції**

Конструкція if ... else

```
if (condition) {
    statements
}

if (condition) {
   if statement
} else {
   else statement
}

if (condition) {
   if statement
} else if (condition) {
   else if statement
} else {
   else statement
}
```

Конструкція switch ... case:

```
switch (expression) {
case value1:
   // run if result of expression equals value1
   break;
case value2:
   // run if result of expression equals value2
   break;
default:
   // run if result not equals none of the values before
      }
```

Тернарний оператор (?:)

condition **?** trueExpression **:** falseExpression;

- **Цикли**

Оператор for

for (initialExpression; condition; incrementExpression or decrementExpression) {
  statements
}

Оператор while

while (condition) {
  statements
}

***Оператор* do ... while**

do {
statements
} **while** (condition);

Цикли для об'єктів і масивів

for (variable in object) {
  statements
}

Цикли for ... in використовуються для обходу властивостей об'єктів, які не є масивами.
Конструкція for ... in перебирає ключі об'єкта.

for (variable **of** array) {
  statements
    }

Цикл for ... of призначений для ітерації за елементами масиву, але на відміну від циклу for ... in при ітераціях використовується значення, а не ключ. Іншими словами for ... of перебирає значення масиву.

Для зчитування інформації із полів вводу на сторінці та виводу на сторінку відповідним тегам задайте ідентифікатори та дотримуйтеся нижченаведеного алгоритму написання функціоналу та виклику події натиснення на кнопку.

```
//  оголошення змінних
let input = document.querySelector("input");

const resultOut = document.querySelector("#result"),
      btn = document.querySelector(".btn");
……
// опис функціоналу
function run() {
let name = “winner”,
…….
  // вивід результату
resultOut.innerHTML = `<strong> ${name }</strong>
                              переміг з ${input.value } балами`;
}
  // виклик події
btn.addEventListener("click", run);
```

Детальніше про використані методи та властивості можна прочитити у таких статтях з [3]:

1. Для роботи з математичними формулами викликати вбудований об’єкт Math. (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)
2. Для роботи з текстом викликати вбудований об’єкт String (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String).

3. Активно використовуйте шаблонний рядок (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals?retiredLocale=uk).
4. Для звернення до об'єктів DOM використовувати метод **`querySelector()`** (https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector)
5. Для виведення інформації на вебсторінку можна користуватись властивістю **`innerHTML`** (https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML).
6. Для генерації подій використовувати метод **`addEventListener()`** (https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener)

Завдання та хід виконання

1. Засобами HTML і CSS створити вебсторінку, на якій розмістити:
   - навігайну панель з переліком завдань;
   - блоки реалізації завдань;
2. Назви елементів навігаційної панелі мають відповідати назвам завдань.
3. Блоки завдань зв'язати з відповідними назвами навігаційної панелі. Навігаційна панель має бути закріпленою.
4. На навігаційній панелі має бути лого, при наведені на яке з'являється вікно із зображенням і короткою інформацією про автора сторінки.
5. Стилізувати сторінку за власним уподобанням.
6. Якщо реалізація завдання громіздка, то реалізовувати кожне завдання в окремому script файлі.
7. Виберіть з переліку завдань принаймні 1 для реалізації.
8. Самостійно запропонуйте і реалізуйте ще 1-2 завдання на інші теми. Тематика не обмежується.

9. Усього має бути реалізовано не менше 3 завдань.

10.Передбачити контроль введення вхідних даних.

11.Оформіть лабораторну роботу.

12.Захистіть лабораторну роботу до зазначеного строку, вказаного на Moodle.

13.Завантажте архів роботи на Moodle. Можна розміщувати саму лабораторну роботу на GitHub.com чи на власному Google диску, а у текстовому звіті надавати посилання на неї.

Зауваження. Кількість балів залежить **від складності завдання і повноти реалізації.**

Перелік пропонованих завдань:

Допомога школяреві:

1. Реалізувати калькулятор.
2. Перевірити введене натуральне число на подільність на 2, 3, 5, 9, 10. Пояснення ознак подільності виводити.
3. Знайти найбільший спільний дільник для пари натуральних чисел.
4. Знайти найменше спільне кратне для пари натуральних чисел.
5. Визначити кількість розрядів введеного цілого числа.
6. Вивести число у зворотному порядку цифр.
7. Знайти суму і добуток цифр числа, що вводиться.
8. Знайти дійсні корені лінійного та квадратного рівняння і вивести їх на екран, якщо вони є. Якщо коренів немає, то вивести повідомлення про це.
9. За двома вхідними даними прямокутного трикутника вивести інші розміри лінійних величин та кутів.
10. За трьома довжинами відрізків з'ясувати чи можна побудувати трикутник. Якщо так, то вказати його тип.
11. Залежно від вибору користувачем виду фігури обчислити площу кола, прямокутника або трикутника за вхідними даними.

12. Перевести одні величини у інші за вибором (кілометри, метри, сантиметри, дециметри, міліметри).

13.Перевести одні величини у інші за вибором (байти, кілобайти, мегабайти, гігабайти, терабайти).

14.Вивести на екран стільки елементів ряду Фібоначчі, скільки вказав користувач.

15.Вивести на екран факторіал числа, що вказав користувач. Крім того, перевірити парність чи непарність числа і вивести подвійний факторіал.

16.Ввести набір чисел. З'ясувати скільки разів зустрічається певна цифра серед них.

17.Ввести набір чисел. З'ясувати чи є це арифметична чи геометрична прогресія. Знайти їхню суму.

18.Серед натуральних чисел, які були введені, знайти найбільше за сумою цифр. Вивести на екран це число та суму його цифр.

19.Знайти середнє арифметичне окремо парних і непарних чисел певного набору.

20.З клавіатури вводиться натуральне число. Знайти його найбільшу цифру.

Допомога фінансисту:

1. Правопис грошової одиниці. Вводиться невід'ємне ціле число. У залежності від суми виводити поряд з числом правильний варіант написання гривні. Наприклад, 241 гривня, 242 гривні, тощо

2. **Суми прописом.** Вводиться невід'ємне число ціле або з двома знаками після коми. Вивести прописом це число. Наприклад, 47 – сорок сім.

3. **Контроль введення даних.** Поступово вводиться набір чисел. Якщо число повторюється, то вивести повідомлення, що таке число вже введено.

4. Усунення повторення даних. Серед набору дат забрати дублюючі. Кожна дата має виводитись лише раз. Узагальнити на усунення дубляжу пар даних (Назва товару; Дата постачання)

5. Повторення даних. Визначити скільки разів присутнє кожне значення у введеному наборі.

Допомога редактору:

1. У введеному тексті забрати зайві пробіли.
2. Обчислити кількість символів у тексті.
3. Обчислити кількість введених слів у тексті.
4. Знайти в рядку зазначений підрядок і замінити його на новий. Рядок, його підрядок для заміни та новий підрядок вводить користувач.
5. Якщо у введеному тексті речення починається з малої букви, замінити на прописну.

Запитання для повторення

1. Де можна застосовувати мову Java Script?
2. Як можна скрипт підключати до розмітки?
3. Як працює скрипт з ключовими словами defer, async?
4. Яка відмінність між змінними, оголошеними ключовими словами var і let?
5. Яка типізація у JavaScrip?
6. Як відбувається приведення типів змінних?
7. Які групи типів є у JavaScrip?
8. Які є види порівняння у JavaScrip?
9. Які умовні оператори є у JavaScrip?
10. Які оператори циклу є у JavaScrip?

Лабораторна робота № 2

Робота з DOM

Короткі теоретичні відомості

Об'єкти

Об'єкти – це складовий тип даних у JavaScript, який містить колекцію властивостей, кожне з яких має пару ключ-значення.

Значеннями можуть бути:

- числа,
- рядки,
- булеві значення,
- функції
- або ж інші об'єкти.

Об'єкти можна створювати за допомогою літералів об'єктів, ключового слова new або функції Object.create ().

```
let object = {};
let newObject = new Object();
let createObject = Object.create();
```

Віддають перевагу першому способу.

```
let flat = {
   name: “3-rooms”
   hall: 20,
   rooms: 70,
   bathroom: 10,
   total: function() {
      return this.rooms + this.hall + this.bathroom;
   };
```

За допомогою оператора delete можна видалити властивість з об'єкта.

Object.keys() – метод, який повертає масив імен властивостей об'єкта.

```
let products = {
```

```
    juice: 'orange',
    fruit: 'banana',
    vegetable: 'carrot'
};

let productsName = Object.keys(products);
console.log(productsName); // ["juice", "fruit", "vegetable"]
```

Object.assign() – копіює передані об'єкти в зазначений цільовий об'єкт і повертає нову копію об'єкта з усіма властивостями. Першим параметром у методі передається цільовий об'єкт, тобто об'єкт, у який будуть скопійовані всі значення. Другий і наступні параметри – це об'єкти, які необхідно скопіювати.

```
let products = {
    fruit: 'banana',
    vegetable: 'carrot'
};

let newProduct = {
    juice: 'orange'
}

let allProducts = Object.assign({}, products, newProduct);
allProducts; // {fruit: "banana", vegetable: "carrot", juice: "orange"}
```

Просте поверхневе копіювання можна здійснювати за допомогою spread оператора.

Наприклад,

```
const user = { name: 'Mykola',  age: 44 };

const copyOfUser = { ...user };
//   рівносильно Object.assign({}, user);
```

За допомогою spread оператора легко розширювати нові об'єкти додатковими даними:

```
const user = { name: 'Mykola', age: 16};
```

```
const newUser = { ...user, student: true, age: 18 };
// додалась нова властивість та змінився вік, оскільки значення властивості, що правіше має більший пріоритет
console.log(newUser); // => { name: "Mykola ',  student: true, age: 25 }
```

Зауваження. Spread оператор можна вставляти й у середині для розширення.

Масиви

Масив – це впорядкована колекція значень. Створити масив можна двома способами:

- за допомогою літерала (конструкція [])
- або ключового слова new.

Створення масиву за допомогою літерала (конструкція []) використовується практично завжди.

```
let array = [];
let newArray = new Array();
let array = ['HTML', 'CSS'];
let newArray = new Array(5);
```

Масиви – це спеціальний тип об'єктів. Тому для елементів (літералів масиву) діє spread оператор.

```
let array1 = [ 'CSS'];
let array2 = [JS', 'TS'];
let newArray = ['HTML', …array1, …array2];
```

Методи масивів

push() додає один або кілька нових елементів у кінець масиву і повертає його нове наповнення.

pop() видаляє останній елемент масиву, зменшує довжину масиву. Повертає видалене значення.

join() перетворює масив у рядок із заданим роздільником. Якщо роздільник не вказати, то масив буде розділений через кому.
split() розділяє рядок на масив за заданим роздільником.
slice() копіює частину масиву від і до (не включаючи це значення). Отже, щоб створити **повну копію масиву**, достатньо застосувати array.slice(0);
splice() універсальний метод, що виконує вставку або видалення елементів масиву.
reverse() сортує масив у зворотному порядку.
indexOf() повертає номер шуканого елемента в масиві. Якщо елемент не знайдений, то поверне -1.
lastIndexOf() виконує те ж що і indexOf (), але пошук починається з кінця масиву.
find() повертає значення першого елемента в масиві, який відповідає зазначеному умові (в функції).
findIndex() діє аналогічно методу find(), але знаходить не значення, а індекс шуканого елемента.

Ітератори масивів

- **Оператори for і for ... of**

```
const list = ['firstItem', 'secondItem', 'thirdItem'];

for (let i = 0; i < list.length; i++) {
   console.log(list[i]);
}
або
for (let value of list) {
   console.log(value);
}
```

- **forEach()** – це метод, який перебирає усі елементи масиву і для кожного викликає callback функцію. Тобто, в цей метод передається callback функція (функція зворотного виклику), яка може на-

бувати три аргументи (поточний елемент масиву, індекс поточного елемента масиву, масив, який перебирається).

```
let list = ['firstItem', 'secondItem', 'thirdItem'];

list.forEach(function(item){
   console.log(item);
});
```

Зауваження. Метод forEach не змінює масив, на якому він застосований.

- **map і filter**

map() - цей метод **створює новий масив**, який складається з результатів викликів callback функції. Відмінною рисою map від forEach є те, що map повертає новий масив і не змінює вихідний масив.

```
let numbers = [1, 2, 3];
let doubleNumbers = numbers.map(function(number) {
   return number * 2;
});
console.log(doubleNumbers); // [2, 4, 6]
console.log(numbers); // [1, 2, 3]
```

filter() – цей метод ***створює новий масив*** з відфільтрованими, за допомогою callback функції, значеннями. Метод працює аналогічно методу map, з однією невеликою відмінністю - у новий масив потраплять тільки ті значення, які пройшли перевірку в callback функції.

```
let list = ['one', 2, 'three', 'four', null, 6];
let newList = list.filter(item => typeof item === 'string');

console.log(newList); // ["one", "three", "four"]
```

- **reduce і reduceRight**

reduce() – метод який обробляє, за допомогою callback функції, кожен елемент масиву, але при цьому результат кожної такої ітерації записується у проміжне значення.
Метод reduce, крім callback функції, набуває ще одного аргументу – початкове значення, у яке і записуватимуться проміжні значення. Сама callback функція також приймає більше аргументів, ніж попередні методи: проміжне значення, елемент масиву, індекс елемента і сам масив.

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

let sum = numbers.reduce(function(result, number) {
   return result + number;
}, 0);
console.log(sum); // 15
```

reduceRight() – працює аналогічним чином, тільки з кінця масиву.

Функції

Функція повинна **виконувати** тільки ту **дію,** яка закладена в її **назві**.

```
function getCurrentYear () {
   let date = new Date ();
   return date.getFullYear ();
}
```

Види функцій

- **Function declaration і expression**

Синтаксис function declaration

```
function funcDeclaration () {
   console.log ( "Function declaration");
}
```

Синтаксис function expression

```
let funcExpression = function (param) {
   console.log ( "Function expression");
```

```
return  param+100;
}
```

Функціональні вирази (function expression) використовуються, коли тіло функції треба привласнити в змінну, наприклад, для передачі функції як параметра в іншу функцію, або зробити її властивістю об'єкта.

function declaration можна викликати як до, так і після її оголошення, проте function expression доступна для виклику після того як оголошена.

- **Анонімна функція**

Анонімна функція - функціональне вираз, який не записується у змінну, у якого відсутнє ім'я.

```
setTimeout (function () {
                    console.log ( "Anonymous function.");
                   }, 2000);
```

Анонімна функція використовується коли відсутня необхідність викликати функцію по імені десь у коді, тобто вона оголошується в місці її використання.

- **Самовикликна функція (IIFE)**

Самовикликна функція (Immediately Invoked Function Expression (IIFE)) - синтаксична конструкція за допомогою якої можна викликати функцію у місці її визначення.

```
(function () {
  statements
} ());
```

Використовується для створення модулів. У таку функцію можна передавати параметри.

```
(function sum (a, b) {
   console.log (a + b);
} (2, 3));  // 5
```

- **Стрілочна функція (arrow function)**

Стрілочні функції (arrow function) - це більш короткий синтаксичний запис звичайних функцій.

```
let arrowFunction = (arguments) => statements;
```

Наприклад,

```
let getSum = (a, b) => a + b;

let getAdd = a => a + 2;
```

DOM

Об'єктна модель документа (Document Object Model – DOM) служить до роботи зі структурою документа. **DOM** – це відображення документа у вигляді сукупності об'єктів, з якою можна взаємодіяти за допомогою JavaScript. Сукупність об'єктів називають **деревом об'єктів** (DOM tree). Кожен елемент DOM дерева називається **нодою** (node).

Доступ до DOM надається за допомогою глобального об'єкта **document**. Об'єкт document репрезентує вхідну точку в DOM структуру.

- Для отримання елементів **html, head і body** об'єкт document містить відповідні властивості:

```
document.documentElement;
document.head;
document.body.
```

- Для доступу до елементів тіла сторінки використовують методи querySelectorAll() і querySelectorl().

elem.querySelectorAll() повертає всі елементи всередині elem, що задовольняють указаному CSS-селектору. Це фактично колекція-масив, тому можна застосовувати ітератори по масиву або спред оператор, щоб перетворити колекцію у масив і використовувати методи масиву.

elem.querySelector() повертає перший елемент, який відповідає цьому CSS-селектору.

- **elem.closest (css)** шукає найближчого пращура, який відповідає CSS-селектору. Сам елемент також включається у пошук.
- **document.links** – робота з посиланнями на сторінці

Наприклад

```
<button class="button"> Run</button>

<script>
   const element = document.querySelector('.button');
...
</script>
```

Робота з атрибутами

Для роботи з атрибутами існує кілька методів. Методи необхідно викликати на елементі, з атрибутами якого необхідно виконати операції.

Методи для роботи з атрибутами:

element.hasAttribute() виконує перевірку на наявність атрибута. Повертає true або false;

element.getAttribute() отримання значення атрибута;

element.setAttribute() установка значення атрибута;

element.removeAttribute() видаляє атрибут.

Наприклад,

```
<button id=”run” hidden>Run</button>

<script>
  const button = document.querySelector ('#run');
  let hasAttr = button.hasAttribute('hidden');

  if (hasAttr) {
    button.removeAttribute('hidden');
```

```
    button.setAttribute('disabled', 'disabled');
    …
}
    let attrValue = button.getAttribute('disabled');
    console.log(attrValue); // 'disabled'

</script>
```

Робота з класами

Для роботи з класами спеціально створені дві властивості **className** і **classList**.

className отримує список класів, які присвоєні елементу, у вигляді рядка.

classList повертає псевдомасив DOMTokenList, який складається з класів, що присвоєні елементу.

Повернувши такий псевдомасив, на ньому можна викликати кілька спеціальних методів:

element.classList.add() додавання нового класу до елементу;

element.classList.toggle() – якщо вказаний клас присутній у елемента, то він буде вилучений, якщо ж такого класу немає, то він буде доданий до елементу;

element.classList.contains() перевіряє, чи містить елемент указаний клас. Повертає true або false;

element.classList.remove() видаляє у елемента зазначений клас.

Наприклад,

```
<header class="header menu"></header>

<script>
  const element = document.querySelector(.header);
  const classes = element.classList;
  console.log(classes);
  classes.add('new', 'basic');
  classes.remove('menu');
```

// DOMTokenList(3) ["header", "new", "basic" value: "header new basic"]

Завдання та хід виконання

1. Продумайте тематику і макети свого проєкту. Використовуйте інструменти чутливого дизайну для вебсторінок. Передбачте сторінку із списком однотипних елементів, розмічених як картковий дизайн, дані яких завантажитимуться у залежності від вибраного фільтру. Тому передбачте панель фільтрів.

2. При завантаженні сторінки мають завантажуватись лише низка зображень із описом. Опис прихований. Стає доступним, якщо його розгорнути. Дані мають братись із масиву, елементами якого є об'єкти.

3. Стилізація інформації має відповідати картковому дизайну Елементи карткових контейнерів повинні центруватись по середині сторінки при будь-якій кількості. Елементи переходять на наступний рядок, якщо за розміром не поміщаються в один.

4. Додатково*. На зображеннях можуть бути розміщенні додаткові елементи, наприклад розміри знижок по середині чи занадто великі тощо. При натисненні або наведенні на зображенні, вони плавно зменшуються, що дає змогу краще розглядіти саме зображення.

5. Після повного завантаження сторінки за деякий час має з'являтись немодальне вікно на пів екрана з пропозицією про підписку повідомлень з вашого сайту з двома кнопками прийняття пропозиції і її відхилення. У разі прийняття при повторному завантаженні сторінки це повідомлення не з'являтиметься. При натисненні кожної із двох кнопок вікно ховається. Можливе короткочасне повідомлення про подяку за приєднання.

6. Передбачте в певному місці появу модального вікна з рекламою. Момент появи реклами продумайте самі. Це може залежати від місця прокрутки сторінки, так і від певного часу перебування на сторінці. На модальному вікні кнопка його закриття спочатку не

активна. Активною вона стає за певний часовий інтервал. Бажано, щоб цей відлік часового інтервалу було видно.

7. Оформіть лабораторну роботу.

8. Захистіть лабораторну роботу до зазначеного строку, вказаного на Moodle.

9. Завантажте архів роботи на Moodle. Можна розміщувати саму лабораторну роботу на GitHub.com чи на власному Google диску, а у текстовому звіті надавати посилання на неї.

Запитання для повторення

1. Об’єкти. Використання об’єктів?
2. Методи та ітератори об’єктів.
3. Spread оператор. Застосування.
4. Масиви. Методи та ітератори масивів.
5. Чи є масив об’єктом? Чи можна застосовувати до нього spread оператор?
6. Функції. Види функцій.
7. Аргументи функції.
8. Звернення до елементів DOM.
9. Методи для звернення до атрибутів тегів.
10. Робота з класами.

Лабораторна робота № 3

BOM. Події. Бібліотеки

Короткі теоретичні відомості

BOM

Об'єктна модель браузера (Browser Object Model – BOM) – це збірне поняття, яке поєднує безліч об'єктів, які у собі мають властивості та методи для роботи з браузером.

Методи для роботи з браузером, які викликаються на об'єкті window:

window.innerWidth – ширина області вмісту вікна браузера.

window.innerHeight – висота області вмісту вікна браузера.

window.outerWidth – зовнішня ширина браузера, у тому числі всі елементи інтерфейсу.

window.outerHeight – зовнішня висота браузера, у тому числі всі елементи інтерфейсу.

window.pageXOffset – число пікселів, на яке документ було прокручено по горизонталі.

window.pageYOffset – число пікселів, на яке документ було прокручено по вертикалі.

Зауваження. Усі перелічені вище властивості доступні лише читання.

Щоб прокрутити сторінку до необхідних координат, існують два методи:

window.scrollTo() – прокручує сторінку до заданих координат;

window.scrollBy() – прокручує сторінку на задану кількість координат.

Cookie і Web Storage

- **Cookie** – дані про користувача, які зберігаються у браузері та передаються на сервер. Властивість cookie викликається на об'єкті document, а не window. Щоб встановити куки необхідно передати у властивість cookie рядок, у якому зазначатимуться властивості зі значеннями, перераховані через точку з комою.

document.cookie = 'username=John; expires=Thu, 20 Dec 2022 12:00:00 UTC; path=/; domain=""; secure';

Якщо властивість expires не вказати, cookie буде видалено після закриття сторінки браузера. Для отримання всіх cookie на поточній сторінці необхідно просто звернутися до cookie як document.cookie.

Максимальний розмір cookie – 4 Кб.

Для видалення та отримання cookie за значенням, стандартних функцій не передбачено, всі вони реалізуються уручну.

- **Web Storage** – місце для локального зберігання даних у браузері.

Іншими словами, технологія Web Storage дозволяє зберігати дані на стороні клієнта, які ніколи не передаються на сервер.

Існують два об'єкти для зберігання даних на стороні клієнта, які належать до об'єкта window:

localStorage – дозволяє зберігати дані локально без обмежень за часом зберігання та можуть бути видалені лише за допомогою JavaScript;

sessionStorage – дозволяє зберігати дані локально протягом однієї сесії (доки не буде закрито вкладку браузера).

Методи для роботи з Local Storage та Session Storage однакові. Тому розглянемо, наприклад, методи з прикладу localStorage:

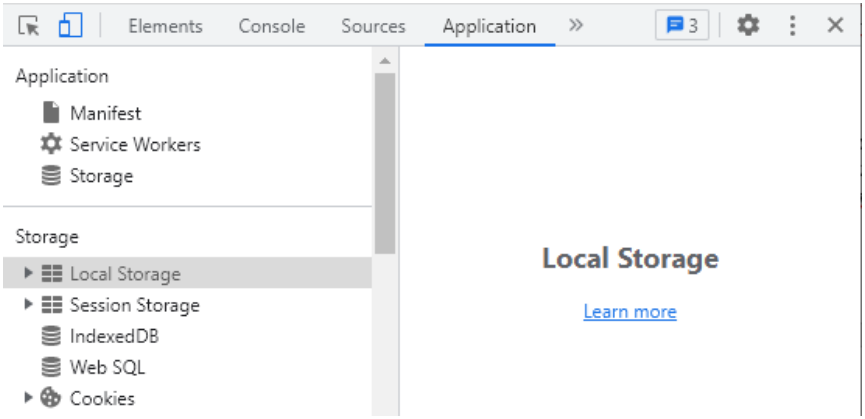
localStorage.setItem() – додати значення до сховища;

localStorage.getItem() – отримання значення зі сховища по ключу;

localStorage.removeItem() – видалити значення по ключу;
localStorage.clear() – очистити сховище.

Максимальний розмір зберігання даних у Web Storage становить 5 Мб.

Cookie та Web Storage можна переглянути і в браузері, якщо запустити DevTools і перейти на вкладку Application.



Події

Подія – це внутрішній механізм JavaScript, який генерує сигнал про те, що відбулась якась дія. Обробником подій є функція, яка виконується після того, як подія відбулася. Способів призначити обробник подій є кілька.

- **Використання HTML атрибуту події.** Цей спосіб полягає в тому, що обробник подій вказується безпосередньо в HTML розмітці, на необхідному HTML елементі, за допомогою спеціальних атрибутів подій. Наприклад, onclick, onchange і т.п.

Наприклад,

```
<input type="text" onfocus="console.log('It works')">
<button onclick="clickHandler()">Button</button>
<span id=”id-span”></span>
```

```
<script>
   function clickHandler() {
     document.querySelector('id-span').innerHTML =
    "Pressed">;
   }
</script>
```

- **Використання DOM-властивості.** У DOM є властивості, які відповідають за події. Називаються вони так само, як і HTML атрибути.

```
<button>Button</button>
<span id=”id-span”></span>
<script>
   const button = document.querySelector('button');

   function clickHandler() {
      document. querySelector ('#id-span').innerHTML =
      "Pressed">;
   }

   button.onclick = clickHandler;
</script>
```

По суті, розглянуті два способи призначення обробника подій ідентичні.

Зауваження. Обидва способи вважаються застарілими і на практиці практично не зустрічаються. Проте, коли треба завантажити одиничний HTML блок із JS, то перший спосіб використовується, бо він простіший у використанні.

- **За допомогою методу addEventListener.** Використання цього методу є сучасним способом призначення події. Сам метод набуває **три параметри**, два обов'язкових, третій опціонний.

Перший параметр - це ім'я події (click, change, mouseover і т.д.).
Другий параметр - функція-обробник події.
Третій параметр - стадія обробки події (спливання або захоплення).
За допомогою цього способу можна додавати кілька обробників подій на один елемент.

```
<button>Button</button>

<script>
   const button = document.querySelector('button');

   function clickHandler() {
      console.log('Pressed');
   }

   function buttonHandler() {
      console.log('Clicked');
   }

   button.addEventListener('click', clickHandler);
   button.addEventListener('click', buttonHandler);
</script>
```

Метод preventDefault() зупиняє стандартну поведінку браузера і дає можливість замість нього виконати якусь свою дію.

Об'єкт event та делегування подій

Об'єкт event – це об'єкт, який містить інформацію про подію, що відбулася.

Суть делегування (Event Delegation). полягає у тому, що якщо є необхідність на кількох елементах встановити однакові обробники події, то замість того, щоб призначати функцію-обробник окремо на кожному елементі, ставиться один обробник для їхнього спільного батька.

За допомогою event.target визначається, на якому елементі відпрацювала подія.

```
<div id="wrapper">
  <button id="first">First </button>
  <button id="second">Second </button>
   <button id="third">Third </button>
</div>

<script>
   const wrapper = document.querySelector('#wrapper');

   function handler(event) {
      console.log(event.target);
      if (eevent.target){……}
      else{…..}
      }

   wrapper.addEventListener('click', handler);
</script>
```

Бібліотеки

Бібліотеки JavaScript неабияк полегшують роботу у написанні коду. Вони створюють програмне середовище, в якому доступно здійснюють схематично та іншою варіацією компонування різноманітної інформації.

- **Бібліотеки роботи з датами.**

Moment.js – потужна бібліотека, яка дає змогу форматувати та маніпулювати датами.

Day.js – це мінімілізована JavaScript бібліотека, яка парсить, валідує, керує, і відображає дати і час для сучасних браузерів, що володіє великою сумісністю з Moment.js.

- **Бібліотеки роботи з 2D графікою.**

Вбудовану графіку можна зображати за допомогою об’єкта canvas. Проте, щоб досягти чутливих різноманітних графіків доці-

льно застосовувати бібліотеки, які є як потужними, так і менш функціональними.

CanvasJS і Chart – це прості у використанні бібліотеки діаграм HTML5 і Javascript.

Бібліотека amChart. поточна версія 5, – сучасна багатофункціональна потужна бібліотека.

- **Бібліотека Jquery** – кросбраузерна бібліотека JavaScript, яка виконує багато типових завдань, для виконання яких потрібно багато рядків коду JavaScript, і об'єднує їх у методи, які можна викликати за допомогою одного рядка коду.

Бібліотека jQuery містить такі функції:

- Маніпуляції HTML/DOM
- Маніпуляції CSS
- Методи подій HTML
- Ефекти та анімація
- AJAX
- Комунальні послуги

Крім того, jQuery має плагіни майже для будь-яких завдань.

Завдання та хід виконання

Частина 1

1. Головну сторінку проєкту, розпочатого у попередній лабораторній роботі, доповніть піктограмами (іконками, спецсимволами, зображеннями або створеними власноруч графічними об'єктами), що є посиланнями на сторінку новин та вікно кошика.

2. Продумайте макет цих елементів проєкту. Макети мають відповідати загальній концепції проєкту. Використовуйте інструменти чутливого дизайну.

3. У вікні формування кошика користувач повинен мати змогу пропонований товар, що поданий у картковому дизайні, сортувати за критеріями:

- назвою;
- ціною (у порядку зростання чи зростання).

А також вибирати товар у певному ціновому діапазоні.

Додаткове завдання*. Можна передбачити одночасний відбір за декількома критеріями. Наприклад, пошук не лише за назвою, а ключовими словами з опису продукту, типом, ціною тощо.

4. Кожен товар можна додати до кошика. На ваш вибір продумайте як оформити при цьому зазначення кількості вибраного товару. Ціна товару не коригується.

Додаткове завдання*. Реалізувати додавання товару до кошика перетягуванням його до піктограми кошик. При цьому з'являється модальне вікно, де за замовчуванням кількість 1. Проте її можна корегувати.

5. У вікні кошика має бути змога коригування кількості, видалення найменування та за потреби повернутись і додати нове найменування. А також виводитись остаточна вартість кошика.

Додаткове завдання*. Передбачити перевірку чи таке найменування вже введено з метою не дублювання найменувань.

6. На сторінці новин має бути стовпчик з із заголовками останніх новин, які завантажуються при відкритті цієї сторінки. Новини мали б оновлюватись по мірі їх надходжень. Розташування новин має бути у хронологічному порядку за датою (формат на вибір) і часом (год і хв). Важливі новини виділятись жирним шрифтом. При натисканні на новині вона розгортається у центральній частині. На мобільних пристроях новина одразу розгортається, а решта посуваються.

Частина 2

7. На підставі вибраних даних за певним критерієм, побудувати принаймні три графічні об'єкти: кругову діаграму, гістограму (стовпчикову діаграму), графік, які б демонстрували популярність товарів у розрізах: тип (назва), кількість (абсолютні чи відносні значення).

8. Дані мають бути масштабовані і змінюватись при змінні даних вибору.

9. Користувач вибирає, який вид графічного об'єкта виводити. Решта приховані.

Частина 3

10. Додайте інтерактивну кнопку навігації вгору. Кнопка має з'являтись, коли сторінка прокручена на 2/3 від висоти браузера.

11. Оформіть лабораторну роботу.

12. Захистіть лабораторну роботу до зазначеного строку, вказаного на Moodle.

13. Завантажте архів роботи на Moodle. Можна розміщувати саму лабораторну роботу на GitHub.com чи на власному Google диску, а у текстовому звіті надавати посилання на неї.

Запитання для повторення

1. Що таке BOM?
2. Як вирахувати позицію курсора у вікні перегляду вебсторінки?
3. Як можна викликати подію? Переваги та недоліки цих способів?
4. Де можна задавати callback функцію при виклику подій? Які функції при цьому можна застосовувати?
5. Що таке делегування подій? Через який об'єкт слід задавати?
6. Як відмінити дію елемента за замовчуванням?
7. Переваги та недоліки застосування бібліотек для написання кодів.
8. Які бібліотеки для роботи з датами можна застосовувати?
9. Як вбудованими засобами JavaScript створити графічний об'єкт?
10. Які бібліотеки для роботи з графікою можна застосовувати?

Лабораторна робота № 4

Обмін даними. Маніпуляції над даними. Валідація форм

Короткі теоретичні відомості

Перевірка даних – це процес, який гарантує, що введені користувачем дані є правильними.

Типовими завданнями перевірки є:

- чи заповнив користувач усі обов'язкові поля?
- чи ввів користувач правильну дату?
- чи ввів користувач текст у числове поле?

Перевірка може бути визначена багатьма різними методами та розгорнута багатьма різними способами.

Перевірка на стороні сервера виконується веб-сервером після того, як введення було надіслано на сервер.

Перевірка на стороні клієнта виконується веб-браузером, перш ніж введення буде надіслано на веб-сервер.

Існують два різних типи перевірки на стороні клієнта:

- **Вбудована перевірка форми** використовує функції перевірки форми HTML5. Для цієї перевірки зазвичай не потрібно багато JavaScript. Вбудована перевірка форми має кращу продуктивність, ніж JavaScript, але її не можна так налаштовувати, як перевірку JavaScript.
- **Перевірка JavaScript** кодується за допомогою JavaScript. Цю перевірку можна повністю налаштувати, але потрібно все створити (або використовувати бібліотеку).

Перевірка обмеження HTML базується на:

- Перевірка обмежень **вхідні атрибути HTML**
- Перевірка обмежень **Псевдоселектори CSS**
- Перевірка обмежень **властивості та методи DOM**

Перевірка за регулярним виразом

Іншою корисною функцією перевірки є pattern атрибуту, який очікує регулярний вираз як своє значення. Регулярний вираз – це шаблон, який можна використовувати для узгодження комбінацій символів у текстових рядках, тому регулярні вирази ідеально підходять для перевірки форми та служать для багатьох інших видів використання у JavaScript.

Регулярні вирази досить складні, Тому треба дивитись документацію. Нижче наведено кілька прикладів, щоб дати основне уявлення про те, як вони працюють.

a – Відповідає одному символу a (не b, не aa тощо).

abc – Збіг з a, потім b, потім c.

ab?c – Збіг з a, необов'язково за якими слідує одиночний b, а потім c. (ac або abc)

ab*c – Збіг з a, за бажанням яких слідує будь-яка кількість bs, за якими слідує c. (ac, abc, abbbbbc, тощо).

a|b – Відповідає одному символу, який є a або b.

abc|xyz – Збігається точно abc або точно xyz

Щоб отримати повний список і багато прикладів, зверніться до документації з регулярних виразів.

Перевірка за допомогою JavaScript

Використовувати JavaScript слід, якщо ви хочете контролювати зовнішній вигляд повідомлень про помилки або працювати зі застарілими браузерами, які не підтримують вбудовану перевірку форми HTML.

API перевірки обмежень

Більшість браузерів підтримують API перевірки обмежень, який складається з набору методів і властивостей, доступних у таких інтерфейсах DOM елементів форми:

- HTMLButtonElement(представляє <button>елемент)
- HTMLFieldSetElement(представляє <fieldset>елемент)
- HTMLInputElement(представляє <input>елемент)

- HTMLOutputElement(представляє <output>елемент)
- HTMLSelectElement(представляє <select>елемент)
- HTMLTextAreaElement(представляє <textarea>елемент)

Детальніше про використані методи та властивості можна прочитити у таких статтях з [3]:

1. Для роботи з регулярними виразами https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions
2. Робота з валідацією форм https://developer.mozilla.org/en-US/docs/Web/API/Constraint_validation
3. API перевірки обмежень https://developer.mozilla.org/en-US/docs/Web/API/HTMLButtonElement

JSON формат даних

JSON (JavaScript Object Notation) – це загальний формат для представлення значень та об'єктів. Його опис задокументований у стандарті RFC 4627. JSON підтримує такі типи даних:

Об'єкти { ... }
Масиви [...]
Примітиви:
рядки,
числа,
логічні значення true/false,
null.

Для роботи з даними в цьому форматі використовується вбудований у браузер об'єкт, який називається також JSON. У нього є 2 методи:

• JSON.parse();
• JSON.stringify().

JSON.**parse**(text, [revieverFunction]) – перетворює рядок у JSON форматі на JavaScript сутність (об'єкт, масив, рядок).

```
const jsonData='{"isAvailable":true,
            "list":[{"name": "All","count":19}]
            }';
const data = JSON.parse(jsonData, function (key, value) {
  if (key == "isAvailable") {
  return "Data is Available";
  } else {
  return value;
  }
});
console.log(data);
```

JSON.stringify(data, [filter]) – серіалізує будь-який об'єкт, масив або структуру в формат JSON.

```
const data = {
  id: 12,
  name: "john",
  params: [12,15],
  date: new Date(),
  calc: function() { return 12; }
}
const result = JSON.stringify(data);
console.log(result);
```

У процесі серіалізації відбувається перетворення об'єкта в текстове представлення, причому діють такі правила:

- якщо значення/поля, яке серіалізується, має вбудований метод toJSON(), то використовуватиметься він замість стандартного підходу;
- функції завжди пропускаються;
- не можна серіалізувати об'єкти, які містять посилання на DOM.

Синхронне та асинхронне програмування

Синхронне програмування означає, що весь код виконується рядково і доти, доки попередня операція не буде закінчена, наступна не буде запущена. Сам по собі JavaScript однопотоковий, тобто одночасно може виконуватися тільки одна операція.

Асинхронний підхід означає, що виконання частини коду можна перемістити з основного потоку виконання у так звану чергу виконання після того, як код в основному потоці виконається, відкладена частина коду, з черги, може бути повернена в основний потік і продовжить виконання.

Асинхронний підхід дозволяє виконувати певні операції "у фоні", наприклад, відправляти запит на сервер

Promise

Promise – об'єкт, який використовується для виконання асинхронних операцій та повертає результат у вигляді успішного виконання або помилки.

Promise може бути в одному з таких станів:

• pending – початковий стан, очікування;

• resolved – операція завершена успішно;

• rejected – операцію завершено з помилкою;

• settled – виконано або відхилено, але не перебуває в стані очікування.

```
const enoughSalary = true;

const buyNewPhone = new Promise(
function(resolve, reject) {
if (enoughSalary) {
  const phone = {
  brand: 'Samsung',
  model: 'Galaxy S22 Ultra',
```

```
  color: 'black'
};
resolve(phone);
} else {
   const reason = new Error('Not enough money.');
  reject(reason);
}
}
);
```

Наступну обробку вищенаведених даних можна зробити в методах then та catch. Для обробки успішного виконання використовують then, а помилки – catch.

```
buyNewPhone
.then(function(data) {
console.log(`I've bought ${data.color} ${data.brand} ${data.model}.`);
})
.catch(function(error) {
console.log(error);
});
```

Fetch

Глобальний метод fetch дозволяє, як і XMLHttpRequest, відправляти асинхронний запит через мережу. Перевагою fetch є спрощений синтаксис, він повертає Promise, що дозволяє легше конфігурувати запити.

Метод fetch() – сучасний і дуже потужний. Він не підтримується старими (можна використовувати поліфіл), але підтримується сучасними браузерами.

```
let promise = fetch(url, [options])
```

Без options це простий GET-запит, що завантажує вміст на адресу url. Браузер відразу починає запит і повертає проміс, який зовнішній код використовує для отримання результату.

Процес отримання відповіді зазвичай відбувається у два етапи.

По-перше, promise виконується з об'єктом убудованого класу Response як результат, як тільки сервер надішле заголовки відповіді.

- Для надсилання GET запиту достатньо вказати URL.

```
fetch('https://jsonplaceholder.typicode.com/posts?userId=1')
  .then((response) => {
    return response.json();
  })
  .then((data) => {
    console.log(data);
  });
```

- Для надсилання запиту POST слід визначити другий параметр fetch.

```
const options = {
  method: 'post',
  headers: {
    'Content-type': 'application/x-www-form-urlencoded; charset=UTF-8'
  },
  body: 'title=foo&body=bar&userId=1'
}

fetch('https://jsonplaceholder.typicode.com/posts', options)
  .then((response) => {
    return response.json();
  })
  .then((data) => {
```

```
    console.log('Request succeeded with JSON response', data);
  })
  .catch((error) => {
    console.log('Request failed', error);
  });
```

Async/await

Конструкція async/await дозволяє створювати асинхронні функції. Така функція повертає Promise.

```
const getProducts = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(['bread', 'water', 'oil']);
    }, 2000);
  });
};

const buy = (products) => {
  const msg = `You bought: ${products.length} products.`;
  return Promise.resolve(msg);
}

async function order() {
  let products = await getProducts();
  let orderMessage = await buy(products);

  return orderMessage;
};

order()
  .then((response) => {
    console.log(response);
  });
// You bought: 3 products.
```

Завдання та хід виконання

Частина 1

1. Головну сторінку проєкту, розпочатого у попередніх лабораторних роботах, доповніть піктограмами (іконками, спецсимволами, зображеннями або створеними власноруч графічними об'єктами), що є посиланнями на сторінку чи вікно авторизації та реєстрації. Форма реєстрації має обов'язково мати поля логіну, паролю та його повторення. Інші поля за вашим уподобанням.

2. Продумайте макет цих елементів проєкту. Макети мають відповідати загальній концепції проєкту. Використовуйте інструменти чутливого дизайну.

3. Перед відправкою на сервер даних має пройти повна валідація на наявність заповнення обов'язкових даних та правильність заповнення їх. Наприклад, пароль має мати не менше певної кількість символів. Користувач має розуміти заповнення яких полів обов'язкове.

Додаткове завдання*. Можна передбачити щоб пароль мав символи з набору обов'язкових символів.

4. Пропишіть перевірку на збіг введеного паролю та його повторення.

5. У разі не проходження валідації на етапі реєстрації користувач має бачити які саме неточності при заповненні даних він допустив.

6. У разі не проходження валідації на етапі авторизації користувач, навпаки, не має бачити які саме неточності при заповненні даних він допустив. Лише з'являється інформаційне повідомлення про не проходження авторизації.

7. В обох випадках у разі проходження валідації має з'являтись інформаційне повідомлення про її успіх.

8. Замовлення з кошика може відправляти лише зареєстрований користувач. Допишіть повідомлення у разі не авторизації його і перенаправлення на авторизацію/реєстрацію

Частина 2

9. Установити локальний сервер.

10.Дані товарів, які завантажуються на сторінку, подати у JSON форматі. Надалі використовувати їх, розпарсивши у подальшому.

Додаткове завдання*. Доповнити кожен елемент товару/пропозиції додатковою властивістю популярності для фіксації кількості їх вибору користувачами. За даними їхньої популярності відобразити для адміністратора діаграми.

11.Аналогічно й дані користувачів і адміністратора. Дані адміністратора є з самого початку. Він не реєструється. Продумайте як розрізнити звичайного користувача й адміністратора.

12.Редагувати, видаляти, додавати нові товари/пропозиції може лише адміністратор. Користувач може лише переглядати їх.

13.Створіть відповідні запити для CRUD операцій для:

- реєстрації/авторизації;
- маніпулювання даними товарів/пропозицій.

14.Додаткове завдання*. Створити імітацію особистого кабінету користувача, де він бачить свої особисті дані. Може змінювати пароль і контакту інформацію і можливо бачить зроблені попередні замовлення.

15.Оформіть лабораторну роботу.

16.Завантажте архів роботи на Moodle. Можна розміщувати саму лабораторну роботу на GitHub.com чи на власному Google диску, а у текстовому звіті надавати посилання на неї.

17.Зареєструйтесь на хостингу за уподобанням, наприклад. на netlify.com

18. Розмістіть та ресурсі свій проєкт. Продемонструйте роботу викладачу.

19. Захистіть лабораторну роботу та розмістіть на сторінці курсу архів проєкту та посилання на хостинг.

Запитання для повторення

1. Які види валідації є?
2. Для цього використовувати регулярні вирази?

3. Що таке асинхронний код і де його можна застосовувати?
4. Що таке promise?
5. Як працюють promise?
6. Що таке fetch?
7. Призначення та порядок роботи async/await.
8. Організовувати get- post- запити?

Література

1. Освітня програма «Інформаційні технології та управління проєктами першого (бакалаврського) рівня вищої освіти» [Електронний ресурс] – Режим доступу : https://mathmod.chnu.edu.ua/media/1721/op_knbak_2021.pdf
2. Освітня програма «Системний аналіз першого (бакалаврського) рівня вищої освіти» [Електронний ресурс] – Режим доступу : https://mathmod.chnu.edu.ua/media/1723/ op_sabak_2021.pdf
3. JavaScript. [Електронний ресурс] – Режим доступу : https://developer.mozilla.org/en-US/docs/Web/JavaScript
4. JS підручник. [Електронний ресурс] – Режим доступу : https://w3schoolsua.github.io/html/index.html
5. Сучасний підручник з JavaScript. [Електронний ресурс] – Режим доступу : https://uk.javascript.info/
6. David Flanagan. JavaScript: JavaScript: The Definitive Guide, 7th Edition by David Flanagan Released May 2020 Publisher(s): O'Reilly Media, Inc. ISBN: 9781491952023. [Електронний ресурс] – Режим доступу : https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/
7. JavaScript. Notes for Professionals. [Електронний ресурс] – Режим доступу : https://books.goalkicker.com/JavaScriptBook/
8. John Dean. Web Programming with HTML5, CSS, and JavaScript. [Електронний ресурс] – Режим доступу : https://www.pdfdrive.com/web-programming-with-html5-css-and-javascript-e187657772.html

9. Refactoring: Improving the Design of Existing Code by Martin Fowler, Released November 2018, Publisher(s): Addison-Wesley Professional, ISBN: 9780134757681. [Електронний ресурс] – Режим доступу : https://www.oreilly.com/library/view/refactoring-improving-the/9780134757681/
10. JQuery підручник. [Електронний ресурс] – Режим доступу : https://www.w3schools.com/jquery/default.asp

Інформаційні ресурси

1. Український вебдовідник. [Електронний ресурс] – Режим доступу : https://css.in.ua/
2. Основи, інструменти, оновлення та приклади з web-розробників Google [Електронний ресурс] – Режим доступу : https://www.html5rocks.com/en/
3. Сайт розробників GOOGLE. [Електронний ресурс] – Режим доступу : https://www.developers.google.com
4. JQuery. [Електронний ресурс] – Режим доступу до ресурсу: https://jquery.com/
5. JQuery UI. [Електронний ресурс] – Режим доступу : https://jqueryui.com/

Навчальне видання

Укладач:

Готинчан Тетяна Іванівна

Вебтехнології та вебпрограмування

Методичні вказівки та завдання до лабораторних робіт

Відповідальний за випуск **Черевко І.М.**
Комп'ютерний набір **Готинчан Т.І.**