Міністерство освіти і науки України
Житомирський державний університет імені Івана Франка

ЛАБОРАТОРНИЙ ПРАКТИКУМ З WEB-ТЕХНОЛОГІЙ ТА WEB-ДИЗАЙНУ

Житомир
Вид-во ЖДУ ім. І. Франка
2024

УДК 004.774(076.5)
Л12

Укладачі: О. О. Мосіюк, В. С. Прухницький

Затверджено на засіданні вченої ради Житомирського державного університету імені Івана Франка,
протокол № 23 від 24.12.2024.

Рецензенти:

Олександр МІЦА – завідувач кафедри інформаційних управляючих систем та технологій ДВНЗ «Ужгородський національний університет», доктор технічних наук, професор

Оксана НАКОНЕЧНА – доцент кафедри інформаційних технологій Одеського державного аграрного університету, кандидат технічних наук

Олена УСАТА – кандидат педагогічних наук, доцент, завідувач кафедри комп'ютерних наук та інформаційних технологій Житомирського державного університету імені Івана Франка

Л12 Лабораторний практикум з web-технологій та web-дизайну / уклад.: Мосіюк Олександр, Прухницький Віталій. Житомир: Вид-во ЖДУ ім. Івана Франка, 2024. 72 с.

Лабораторний практикум з web-технологій та web-дизайну передбачає формування у студентів комплексних знань та умінь з використання технологій розробки сучасних вебсайтів за допомогою інструментарію HTML, CSS, JavaScript.

УДК 004.774(076.5)

ЗМІСТ

ВСТУП

Вебтехнології є одним ключових напрямів сучасного цифрового ринку програмного забезпечення. Вони задають тренд взаємодії користувачів з Internet-простором. Такі технології охоплюють широкий спектр інструментів і методів для створення інтерфейсів відповідних застосунків, їх функціоналу й оптимізації. Дизайн поєднує в собі технічні знання, креативність та користувацький досвід, роблячи взаємодію з сайтами максимально зручною й ефективною.

HTML (HyperText Markup Language) є базовою технологією, що відповідає за структуру вебсторінок. Він дозволяє розміщувати текст, зображення, посилання, таблиці та інші елементи. Паралельно з ним використовуються CSS (Cascading Style Sheets), які відповідають за оформлення та візуальний стиль сторінок. Саме CSS визначає кольори, шрифти, розташування блоків і адаптивний дизайн вебресурсів.

JavaScript – це мова програмування, завдяки якій реалізується динамічна поведінка сторінок відповідних вебзастосукнків. За його допомогою можна додавати інтерактивні елементи та анімації, здійснювати обробку подій, виконувати валідацію форм, розміщувати динамічний контент тощо. У свою чергу сучасні фреймворки, такі як React, Angular й Vue.js, значно спрощують розробку складних односторінкових застосунків.

Одним із важливих напрямків сучасної веброзробки є адаптивний дизайн, що дозволяє створювати сторінки, які виглядають однаково добре на будь-яких пристроях: від настільних комп’ютерів до смартфонів. Для цього використовуються медіа-запити CSS і адаптивні макети, створені на основі Flexbox чи Grid Layout.

Ще одним важливим аспектом є оптимізація веб-сторінок, яка включає в себе підвищення швидкості завантаження, мінімізацію ресурсів і застосування інструментів для SEO (пошукової оптимізації).

У цьому контексті запропонований лабораторний практикум з веб-технологій і веб-дизайну охоплює базові інструменти та технології створення

вебзастосунків та сторінок сайтів. У практикумі містяться загальні теоретичні відомості, інструкції для виконання лабораторних робіт та завдання для самостійної практики.

Кожна лабораторна робота завершується набором питань для самоперевірки, які допоможуть студентам закріпити отримані знання.

Для виконання завдань рекомендовано використовувати Visual Studio Code і сучасні браузери з інструментами розробника. Студентам дозволяється працювати з іншими середовищами розробки, проте результати лабораторних робіт мають бути завантажені до системи контролю версій Git і розміщені у публічному репозиторії на GitHub.com.

Практичні завдання передбачають застосування HTML, CSS, JavaScript та інструментів адаптивного дизайну для створення структурованих, функціональних і привабливих вебсайтів.

ЛАБОРАТОРНА РОБОТА № 1. Основні правила верстки web сторінок.

Теоретичні відомості

HTML

HTML (абревіатура від HyperText Markup Language) є стандартизованою мовою розмітки, що використовується для створення документів, які відображаються у сучасних браузерах. Завдяки HTML можна розміщувати в мережі інформацію у вигляді заголовків, текстів різного розміру, таблиць, списків, зображень, відео та інших матеріалів. Вона також дозволяє додавати гіперпосилання й інтерактивні форми для передачі і обробки даних на серверах. Розробкою і стандартизацією HTML займається організація World Wide Web Consortium (W3C)[1]. Актуальною версією стандарту є редакція від 19 червня 2024 року[2], яку рекомендовано використовувати для створення вебсайтів. HTML є основою для формування семантичної структури документів у веб-просторі.

Представлення інформації на web-сторінці відбувається за допомогою спеціалізованих команд – тегів, які записуються у файлі із розширенням .html або .htm. Всі теги поділяються на дві великі групи.

Парні теги – інструкції, що мають команди, які позначають початок та кінець, у той час як одинарні теги задаються однією інструкцією (приклади 1.1). Вони дозволяють задати семантичну структуру документа, для коректного відображення у браузері.

Приклад 1.1.

```
<p> ТЕКСТ</p>
<body> … </body>
<head> … </head>

<br>
<hr>
<link>
```

Приклад 1.2.

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample#3</title>
</head>
```

[1] W3C URL: https://www.w3.org/Consortium/.

[2] HTML. Living Standard — Last Updated 19 June 2024. URL: https://html.spec.whatwg.org/multipage/

```
<body>
  <!-- Тут розміщується основний вміст документу-->
</body>
</html>
```

Приклад 1.2 демонструє структуру HTML документа.

У першому рядку міститься обов'язкова декларація `<!DOCTYPE html>`, яка вказує браузеру на який стандарт орієнтуватися при відображенні інформації. На даний момент ця інструкція вказує, що використовується специфікація HTML 5.

Тег `<html></html>` визначає основну частину сторінки та містить два вкладених контейнери `<head></head>` та `<body></body>`. У першому контейнері задається інформація призначена для пошуковиків, браузерів, завантаження додаткових елементів оформлення (зокрема каскадних таблиць стилів) тощо. Між відкритим та закритим тегами `<body></body>` подається всі матеріали, які відображатимуться у браузері.

Загалом всі теги документа розділяють на такі великі категорії.

Metadata content[3] – метадані, які необхідні для браузерів, систем пошуку та SEO оптимізації.

Flow content – теги потокового контенту, які дозволяють задати ключові елементи web-сторінки.

Sectioning content – схожі за структурою до попередньої категорії, але обов'язковою умовою є наявність заголовку (наприклад тег `<article>`).

Heading content – теги заголовків.

Phrasing content – інструкції цієї категорії дозволяють представляти текстовий матеріал на web-сторінці (наприклад тег `<article>`).

Embedded content – сторонні вбудовані елементи, найчастіше це зображення.

Interective content – всі елементи сторінки, які безпосередньо будуть взаємодіяти із користувачем.

Також треба зауважити, що теги можуть належати до декількох категорій. Наприклад тег `<scripts>`, який дозволяє розмістити код JavaScript або підключити сторонній js файл у HTML документ дозволяється розміщувати як метадані у контейнері `<head></head>` так і разом із потоковими тегами у тілі самого документа `<body></body>`. На рисунку 1.1 наглядно демонструються основні категорії тегів HTML.

[3] Назви категорій подаються англійською мовою так як вони представлені у оригіналі специфікації.

Перш ніж перейти до опису основних найбільш використовуваних тегів, розкриємо, що являє собою сементична верстка. Отже, *семантична верстка* передбачає використання тегів тільки у відповідності до їхнього призначення, що дозволяє пошуковим системам визначати тип інформації закладеної у цих тегах.

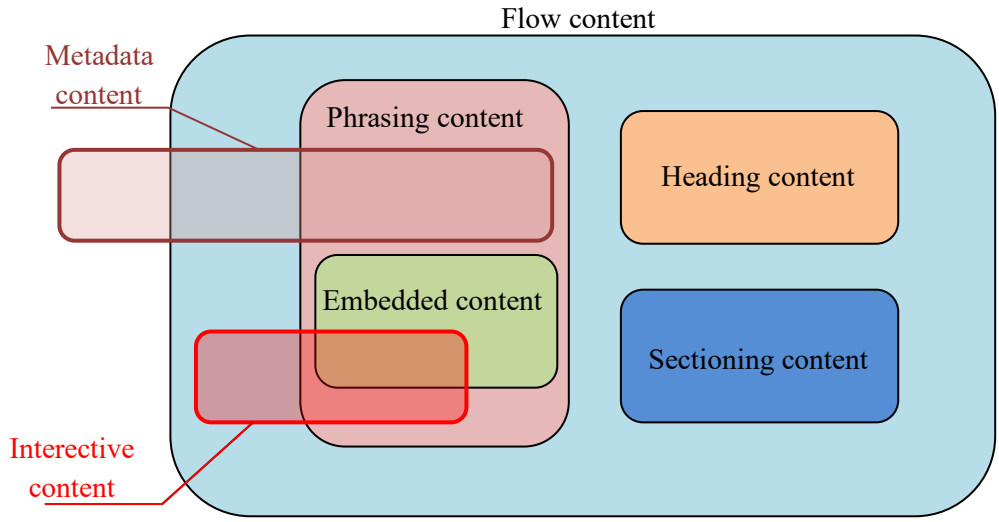


Рис. 1.1. Співвідносність категорій тегів HTML

Тепер розглянемо більш детально основні інструкції (семантичні теги), які формують скелет HTML документу.

`<header></header>` - умовна “шапка” сторінки, а фактично вступний контент, який, типово, містить логотип, основну навігацію, форму пошуку, спеціалізоване випадаюче меню, кнопки входу і реєстрації, тощо.

`<main></main>` - тег, який містить у собі унікальний контент сторінки. Найчастіше він відповідає головній тему документа.

`<footer></footer>` - умовний “підвал” сторінки, що передбачає розміщення інформації про власника сайту, його авторське право, основну навігацію та інші важливі речі.

`<nav></nav>` - задає основні елементи навігації по сайту.

<section></section> - частина сторінки, яка тематично групує вміст і передбачає обов’язкової наявності заголовку.

`<article></article>` - частина сторінки, яка може використовуватися для незалежного розповсюдження або ж повторного використання. Наприклад запис у блозі, картка товару.

`<aside></aside>` - частина сторінки, вміст якої опосередковано пов’язують з контентом сторінки (бічна панель, рекламний банер, додаткове меню навігації або фільтрації контенту тощо).

`<h1></h1>...<h6></h6>` - теги заголовків. Всього існує шість рівнів.

`<p></p>` - блок тексту, абзац. Блоковий тег.

 - рядковий тег, який дозволяє виокремлювати інформацію або інші рядкові теги з метою стилізації.

Ряд тегів можуть використовуватися як для задання певних структурних елементів сторінки або ж використовуватися напряму за призначенням, зокрема до таких тегів можна віднести інструкції, що задають списки.

 - unordered list - невпорядкований список.

 - ordered list - впорядкований список.

Кожен елемент списку має визначатися парним тегом .

<dl></dl> - definition list - список типу термін - визначення. Для терміну використовується тег <dt></dt>, а визначення описується тегом <dd></dd>.

Також часто використовуваними тегами є інструкції, які описують інтерактивні елементи або завантажують графічний контент.

 - створюємо гіперпокликання. Обов'язковий атрибут href задає шлях до ресурса.

<button></button> - тег, який створює кнопку.

 - одинарний тег, який дозволяє завантажувати графічний контент та має два обов'язкові атрибута src - задає шлях до зображення і alt - дозволяє вивести його опис, за умови, що зображення не було відображено на сторінці.

Атрибути тегів

Щоб розширити можливості тегів і більш гнучко керувати вмістом контейнерів застосовуються атрибути тегів, які розштрюють їх функціональність або ж навпаки – інструкції HTML повністю втарчають свою сутність та функціональність. Розглянемо найпоширеніші із них.

Атрибут class – задає зовнішнє оформлення тега. Якщо необхідно застосувати декілька селекторів типу «клас», то вони перераховуються у лапках через пропуск (пробіл).

Атрибут id – діє схожим чином як і попередній, але має значну відмінність. На одній сторінці має зустрічатися тільки один раз селектор id із унікальним ім'ям.

Окрім цих двох найпоширеніших атрибутів існують і спеціалізовані. Вони можуть відноситися до певного типу (текст, число, шлях до файлу тощо), який обов'язково повинен враховуватися при записі значення атрибута. Наприклад тег <img>, що додає на web-сторінку зображення, а його атрибут width задає ширину зображення в пікселях, height – його висоту, src – шлях до файла малюнка, а атрибут alt додає опис до зображення. Для тега

<a> важливим є наявність атрибута href, що вказує посилання на документ у мережі Internet.

Значення атрибута задається у подвійних лапках.

Cascade Style Sheets

CSS (скорочення від англійських слів Cascade Style Sheets або каскадні таблиці стилів) властивості задають для web-документа його оформлення і поведінку блоків. Саме форматування сторінки за допомогою каскадних таблиць стилів забирає левову частку процесу верстки.

Властивості CSS умовно можна поділити на такі великі групи властивостей:

- задають позиціонування об'єктів;
- пов'язані із встановленням розмірів і відступів;
- керуванням «потоком» документа;
- форматування тексту;
- декоративне оформлення елементів;
- елементи анімації та динамічних ефектів.

Окремо варто виділити псевдоелементи та псевдокласи.

Синтаксис CSS є достатньо простим і його умовно можна подати так, як показано у прикладі 1.3.

Приклад 1.3.

```
селектор {
    властивість: значення;
    властивість: значення;
....
    властивість: значення;
}
```

Селектор – це правило, яке визначає стиль та поведінку HTML-елемента, до якого воно застосовується. Опис стилю, зрозумілий для браузера, записується між відкритими та закритими фігурними дужками у форматі пар «властивість: значення» Кожна властивість відокремлюється від значення двокрапкою, а завершення опису кожної властивості позначається крапкою з комою.

Для коментування частини коду використовується наступний запис (приклад 1.4).

Приклад 1.4.

```
/* Це коментар у CSS*/
```

Стилі до HTML сторінки можуть підключатися наступними способами.

Рядковий (inline) - стилі задаються в атрибуті style

Вбудований (embedded) - стилі задаються всередині тегу *<style></style>* у шапці документа

Зовнішній (external) - стилі задаються в окремому файлі з розширенням .css.

До основних видів селекторів відносять.

Селектор тегу - назва тегу (наприклад `section`)

Селектор класа - записується за допомогою крапки на початку

Селектор id - записується за допомогою знаку `#`

Селектор атрибуту - задається записом на початку знаків `[]` і назви атрибуту. Наприклад `[type = "submit"]`.

Селектор нащадка записується через знак більше. Наприклад: `section > h2`.

При написанні коду важливо пам'ятати про каскадність. Каскадність є механізмом, який керує кінцевими значеннями властивостей елемента, якщо до нього застосовується кілька CSS правил. Тому завжди треба брати до уваги такі правила.

Якщо до елемента застосовується кілька правил, їх властивості комбінуються.

Якщо правила містять однакові властивості з різними значеннями, то вони конфліктують.

Для CSS також важливим є поняття *специфічності* (ваги селектора, що розраховується браузером). Якщо до елементу потрібно застосувати властивості з різних правил, а існують однакові властивості, застосовується значення властивості з правила, що має більшу специфічність селектора. Окрім цього для каскадних таблиць стилів реалізовано механізм успадкування, за допомогою якого значення певних властивостей передаються від предка до його нащадка.

Завдання до лабораторної роботи

1. Зверстати HTML сторінку відповідно до Вашого варіанту.
2. Структура блоків, їх розміри та розміщення мають відповідати зразку.
3. Колір блоків має відповідати кольору цих елементів на зразка.
4. Текст повинен бути таким, як представлено на рисунку.
5. Каскадна таблиця стилів має бути розміщена в окремому файлі та підключатися до HTML файлу.
6. Всі фали мають бути у репозиторії на сайті github.com або gitlab.com із назвою **Прізвище_NNg_Lab1** **або** бути надіслано на електронну пошту викладача у **ZIP архіві**.
7. Посилання на репозиторій слід надіслати на пошту викладача.

Варіанти

1. Варіант URL: Var_1.png
2. Варіант URL: Var_2.png
3. Варіант URL: Var_3.png
4. Варіант URL: Var_4.png
5. Варіант URL: Var_5.png
6. Варіант URL: Var_6.png

Питання для самоперевірки

1. Що таке HTML і як він використовується у верстці web сторінок?
2. Яка роль CSS у верстці web сторінок?
3. Що таке HTML-теги і як вони впливають на структуру web сторінки?
4. Поясніть різницю між внутрішніми, зовнішніми та вбудованими CSS-стилями.

Довідкові матеріали

1. Мосіюк О. О. WEB-технології. Частина 1. Верстка. – Житомир: Вид-во ЖДУ ім. Івана Франка, 2020. – 56 с. URL:chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://eprints.zu.edu.ua/32361/1/Web_ost.pdf
2. HTML. Основи. URL: https://slides.com/mos_xandr/deck-4-7
3. CSS. Основи. URL: https://slides.com/mos_xandr/deck-6-9?authuser=0
4. Блочна верстка. URL: https://slides.com/mos_xandr/deck-4-7-11?authuser=0
5. ITDVN Верстальник сайтів URL: https://itvdn.com/ua/specialities/html-coder

ЛАБОРАТОРНА РОБОТА № 2. Форматування тексту за допомогою CSS.

Теоретичні відомості

CSS властивості для форматування тексту

Шрифт є упорядкованою множиною літер певного типу для відображення текстової інформації. Розрізняють такі його основні типи: шрифти з засічками (`serif`), шрифти без засічок (`sans-serif`), шрифти однакової ширини (`monospace`).

Шрифти у CSS задаються властивістю `font-family`, але якщо це не локальний і не поширений шрифт, то у такому випадку їх наобхідно завантажити та вже тоді посилатися.

Декорування тексту відбувається шляхом використання властивості `text-decoration`. Вона може набувати наступних значень `overline, line-through, underline, none`. Завдяки їм дозволяється реалізувати різні варіанти підкреслення, перекреслення і закреслення.

Вирівнювання тексту по горизонталі задається властивістю `text-align: left (center, right)`.

Зміни регістру та форми написання тексту задають завдяки властивісті `text-transform: uppercase (lowercase, none, capitalize)`.

Відстані між літерами задає властивість `letter-spacing`, між словами визначається за допомогою `word-spacing`, а якщо треба показати як буде відображатися текст, якщо він не вміщується у один рядок (дозволяє або забороняє перенесення), то тоді задають властивість `white-space`.

Величина міжрядкового інтервалу задається за допомогою властивості `line-height`. Рекомендовано використовувати відносне значення.

Досить часто використовуються властивості `font-size` (задає розмір шрифту), `font-weight` (товщина накреслення символів), `font-style` (тип накреслення).

Колір тексту задається за допомогою властивості `color`, а імітація тіні, яку ніби відкидає символи тексту, вказуються за допомогою інструкції `text-shadow: зміщ.по_осі_x зміщ.по_осі_y ступінь_розмиття колір`;

Завдання до лабораторної роботи

Зверстати основну сторінку сайта у відповідності до запропонованої схеми (варіанту). Він повинен відповідати наступним вимогам.

1. Розміри блоків та відступів мають бути такими, як на схемі.

2. Ширина основного блока 980px (за винятком варіантів, у яких вона вказана на рисунку).

3. Необхідно підібрати зображення у відповідності до тематики сайта.

4. Елементи меню мають змінюватися при наведенні на нього кнопки миші.

5. Форми введення мають бути замінені на візуально подібні блоки.

6. Колір шрифту, його розмір має наближатися до оригіналу, шрифт - Georgia або Sans Serif.

7. Забороняється використовувати таблиці.

8. Всі фали мають бути у репозиторії на сайті github.com або gitlab.com із назвою Прізвище_NNg_Lab2 або бути надіслано на електронну пошту викладача у ZIP архіві.

9. Посилання на репозиторій слід надіслати на пошту викладача.

Тематика варіантів

1. Варіант 1. Спорт.
2. Варіант 2. Фінанси.
3. Варіант 3. Захист природи.
4. Варіант 4. Заклад швидкого харчування.
5. Варіант 5. Курси підготовки до ЗНО
6. Варіант 6. Офіційний сайт школи.

Варіанти

1. Варіант URL: Lab2_V1.png
2. Варіант URL: Lab2_V2.png
3. Варіант URL: Lab2_V3.png
4. Варіант URL: Lab2_V4.png
5. Варіант URL: Lab2_V5.png
6. Варіант URL: Lab2_V6.png

Питання для самоперевірки

1. Що таке властивість `font-family` і як її використовувати для зміни шрифтів на web сторінці?

2. Як працює властивість `font-size`, і які одиниці вимірювання можуть бути використані для задання розміру шрифту?

3. Що таке `line-height` і як вона впливає на відстань між рядками тексту?

4. Поясніть, як використовувати властивості `font-weight` і `font-style` для зміни товщини та стилю тексту.

5. Як можна змінити колір тексту за допомогою властивості `color`?

Довідкові матеріали

1. Блочна верстка. Частина 2. URL: https://slides.com/mos_xandr/deck-6-9-12?authuser=0
2. HTML Форматування тексту URL: https://w3schoolsua.github.io/html/html_formatting.html#gsc.tab=0
3. HTML Зображення URL: https://w3schoolsua.github.io/html/html_images.html#gsc.tab=0
4. CSS :hover Selector URL: https://www.w3schools.com/cssref/sel_hover.php
5. HTML <a> Tag URL: https://www.w3schools.com/tags/tag_a.asp

ЛАБОРАТОРНА РОБОТА № 3. Робота із формами в HTML5 та CSS3.

Теоретичні відомості

Теги HTML та властивості CSS для роботи з формами

Створення форм для введення та передачі даних від користувача на сервер є важливою частиною верстки сучасних веб-сторінок. Це дозволяє організувати обробку даних і забезпечити зворотний зв'язок. Для створення форм використовують тег `<form> ... </form>`.

Ключовими атрибутами цього тега є такі: `accept-charset`, `action`, `autocomplete`, `enctype`, `method`, `name`, `novalidate`, `target`. Розглянемо їх значення докладніше:

`accept-charset` – задає кодування, в якій сервер зможе приймати та обробляти дані;

`action` – адреса скрипта, який обробляє дані форми;

`autocomplete` – умикає автозаповнення полів форми (значеннями, які набуває атрибут є on та off);

`enctype` – вказує способи кодування даних форми (значеннями, які набуває атрибут є application/x-www-form-urlencoded, multipart/form-data та text/plain);

`method` – метод передачі даних протоколу HTTP (значеннями, які набуває атрибут є get та post);

`name` – задає ім'я форми;

`novalidate` – скасовує вбудовану перевірку даних форми на коректність введення;

`target` – ім'я вікна або фрейму, куди скрипт буде завантажувати необхідний опрацьований результат.

Серед них найчастіше використовуються атрибути `action`, `method` та `name`.

Розглянемо більш детально значення атрибута `method`.

Метод передачі `GET` є одним із найбільш поширених і використовується для отримання інформації та передачі даних через адресний рядок. Дані передаються у вигляді пар «ім'я=значення», які додаються до URL після знака питання і розділяються символом «&». Обсяг даних, які можна передати за допомогою `GET`, обмежений 4 кілобайтами.

Метод `POST`, на відміну від `GET`, передає дані безпосередньо у тілі запиту. Це дозволяє відправляти значно більші обсяги інформації. Метод `POST` зазвичай використовується для обробки великих даних, таких як форми на форумах, поштові служби, оновлення баз даних або пересилання файлів.

Основні поля, кнопки, чекбокси та інші елементи, разом із їх підписами, створюються за допомогою тегів: теги, які створюють елементи форми чи спеціальні позначення (<button> … </button>, <input>, <label> … </label>, <legend> … </legend>, <option> … </option>, <output> … </output>, <select> … </select>, <textarea> … </textarea>), теги, що групують елементи (<fieldset> … </fieldset> та <optgroup> … </optgroup>) та виконують спеціальне шифрування (<keygen> … </keygen>). Розкриємо значення кожного із них.

<button> … </button> – створює на web-сторінці кнопки. В середині цього тега дозволяється розміщувати будь-які елементи HTML, у тому числі зображення. Використовуючи CSS стилі, дозволяється змінити зовнішній вигляд кнопки шляхом зміни шрифту, кольору фону, розмірів або інших параметрів.

Серед важливих атрибутів тега треба виділити такі.

accesskey – доступ до елементів форми за допомогою гарячих клавіш;

autofocus – вказує, що кнопка отримує фокус після завантаження сторінки;

disabled – блокує доступ і зміни елемента;

form – зв’язує кнопку із формою за її ідентифікатором (задається атрибутом name у формі). Такий зв'язок необхідна тоді, коли кнопки не розташовується всередині тега <form>, наприклад, при створенні її програмно.

formaction, formenctype, formmethod, formnovalidate, formtarget – діють аналогічно до атрибутів форми action, enctype, method, novalidate, target. Якщо одночасно вказати для атрибутів форми і їм відповідним у кнопці, то при натисканні на кнопку аналогічні атрибути форми ігнорується, а пріоритетними залишаються ті, які були задані до тега <button> … </button>.

name – задає унікальне ім'я для кнопки;

type – визначає тип кнопки (button –звичайна кнопка, reset – кнопка для очищення введених даних форми та повернення значень в первісний стан, submit – кнопка для відправки даних форми на сервер). За замовчуванням визначається значення submit.

value – значення кнопки, яке буде передано на сервер або прочитано за допомогою скриптів.

Тег <input> призначений для створення текстових полів, різних кнопок, перемикачів і чекбоксів. Основний атрибут тега <input>, що

визначає зовнішній вигляд компонента форми є type. Він дозволяє задавати такі елементи форми як: текстове поле (text), поле з паролем (password), перемикач (radio), прапорець (checkbox), приховане поле (hidden), кнопка (button), кнопка для відправки форми (submit), кнопка для очищення форми (reset), поле для відправки файлу (file) і кнопка із зображенням (image).

Серед важливих атрибутів для цього тега варто назвати такі.

accept – встановлює фільтр на типи файлів, які ви можете відправити через поле завантаження файлів.

accesskey – перехід до елементу за допомогою комбінації клавіш.

autofocus – встановлює фокус в поле форми.

max – верхнє значення для введення числа або дати.

min – нижнє значення для введення числа або дати.

maxlength – максимальна кількість символів дозволених в тексті.

multiple – дозволяє завантажити кілька файлів одночасно.

name – ім'я поля, призначене для того, щоб скрипт, який опрацьовує форму, міг його ідентифікувати.

pattern – встановлює шаблон введення даних у полі.

placeholder – виводить фонову підказку у полі.

readonly – встановлює, що поле не може змінюватися користувачем.

required – указує браузеру, що поле обов'язкове для заповнення.

size – задає ширину текстового поля.

step – крок збільшення або зменшення значень для числових полів.

tabindex – визначає порядок переходу між елементами форми за допомогою клавіші Tab.

value – значення компонента форми.

<label> … </label> – задає підпис елемента форми та дозволяє зв’язати його із текстом підпису. Основними атрибутами для цієї інструкції HTML є: accesskey – задає комбінацію клавіш для активації елемента форми із клавіатури та for – дозволяє зв’язати підпис із компонентом форми, задавши значення його id.

<legend> … </legend> –застосовується для створення заголовка групи елементів форми, яка визначається за допомогою тегу <fieldset> … </fieldset>.

<option> … </option> – визначає окремі пункти списку, що створюється за допомогою контейнера <select>. Ширина списку задається

найбільш довгим текстовим рядком, зазначеним у тегу <option>, а також може змінюватися за допомогою каскадних таблиць стилів.

<output> … </output> – визначає область сторінки, у яку виводиться інформація після опрацювання даних із форми, переважно керується за допомогою скриптів.

<select> … </select> – дозволяє створити список, який розкривається, та список з одним або множинним вибором.

<textarea> … </textarea> – створює область форми, в яку можна вводити кілька рядків тексту. На відміну від тега <input> в текстовому полі дозволяється робити переноси рядків і вони зберігатимуться при відправці даних на сервер.

Окремо розглянемо теги, які дозволяють групувати теги елементів форми. Першим і одним із найпоширеніших у застосуванні є тег <fieldset> … </fieldset> . Це полегшує роботу з формами, які містять велику кількість різного типу полів. Наприклад, один блок може бути призначений для введення текстової інформації, а інший – для чекбоксів або радіокнопок.

<optgroup> … </optgroup> – представляє собою контейнер, усередині якого розташовуються теги <option> об'єднані в одну спільну групу. Особливістю тега є те, що він не є звичайним елементом списку, а виділяється за допомогою жирного накреслення, при цьому всі елементи, що входять до цей контейнер, зміщуються вправо від свого початкового положення.

Тепер наведемо загальні рекомендації, що до коректного оформлення форм для сучасних web-сторінок.

1. Обов’язкова відповідність між вмістом, яке буде вводиться, та типом поля, що буде використано для отримання інформації від користувача.

2. Для тега <form> обов’язковим є наявність атрибутів action, method та name.

3. Для підвищення доступності компонентів форми необхідно описувати поля за допомогою атрибута placeholder.

4. Для підпису поля варто використовувати тег <label> … </label>, який обов’язково прив’язується до поля за допомогою атрибута for. Це необхідно для того, щоб при кліку кнопкою миші по назві відбувалася активація власне самого поля.

5. Для кожного поля id має бути унікальним.

6. Всі радіокнопки та прапорці мають бути названі унікальним іменем.

7. Групи полів мають об’єднуватися за допомогою тега `<fieldset> … </fieldset>` та містити її назву, що задається `<legend> … </legend>`.

Завдання до лабораторної роботи

Зверстати відповідно до варіанту форму введення даних на сайт. Вона повинна відповідати наступним вимогам.

1. Ширина робочої області має становити 1170 px.
2. Необхідно підібрати зображення у відповідності до тематики форми.
3. Елементи меню мають змінюватися при наведенні на нього кнопки миші.
4. Колір шрифту, його розмір має наближатися до оригіналу, шрифт - Tahoma.
5. Забороняється використовувати таблиці.
6. Всі фали мають бути у репозиторії на сайті github.com або gitlab.com із назвою **Прізвище_NNg_Lab3** **або** бути надіслано на електронну пошту викладача у **ZIP архіві**.
7. Посилання на репозиторій слід надіслати на пошту викладача.

Варіанти

1. Варіант URL: Lab3_V1.png
2. Варіант URL: Lab3_V2.png
3. Варіант URL: Lab3_V3.png
4. Варіант URL: Lab3_V4.png
5. Варіант URL: Lab3_V5.png
6. Варіант URL: Lab3_V6.png

Питання для самоперевірки

1. Які основні елементи використовуються для створення форм в HTML5?
2. Що таке атрибут `placeholder` і як його використовувати в полях введення?
3. Поясніть різницю між елементами `input`, `textarea`, `select` і `button` у формах.
4. Які нові типи полів введення з'явилися в HTML5 (наприклад, `email`, `url`, `number`, `date`) і як вони полегшують валідацію даних?.
5. Поясніть, як за допомогою CSS можна стилізувати елементи форми, такі як поля введення, кнопки, випадаючі списки..
6. Валідація форм.

Довідкові матеріали

1. HTML Форми URL: https://w3schoolsua.github.io/html/html_forms.html#gsc.tab=0
2. HTML Атрибути форми URL: https://w3schoolsua.github.io/html/html_forms_attributes.html#gsc.tab=0
3. HTML Елементи форми URL: https://w3schoolsua.github.io/html/html_form_elements.html#gsc.tab=0
4. HTML Типи введення URL: https://w3schoolsua.github.io/html/html_form_input_types.html#gsc.tab=0
5. HTML Атрибути введення URL: https://w3schoolsua.github.io/html/html_form_attributes.html#gsc.tab=0

ЛАБОРАТОРНА РОБОТА № 4. Особливості використання технологій Flexbox та CSS GRID.

Теоретичні відомості

Поняття Flexbox та CSS GRID

Основу будь-якої верстки вебсторінки становить блокова модель. Відповідно до блокової моделі кожному елементу на сторінці відповідає прямокутна область - блок. Для його коректного відображення на сторінці браузер має «знати» про розміри та положення відповідного елемента (рис. 4.1.).

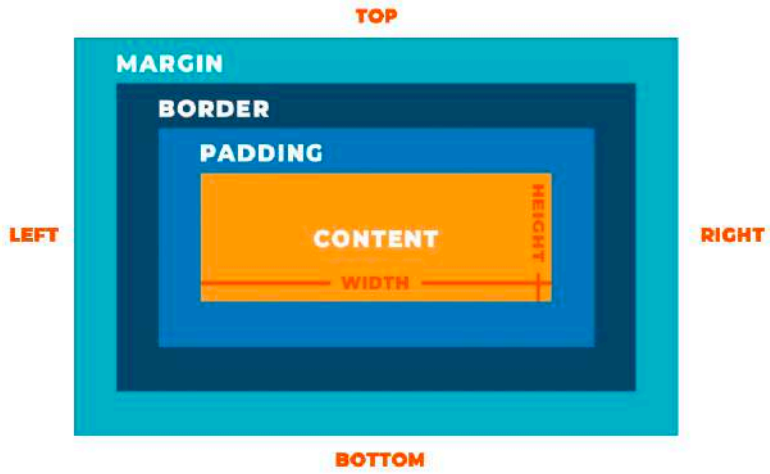


Рис. 4.1. Блокова модель елемента HTML.

На рисунку 4.1 позначено: *content* - місце основної інформації, *padding* - внутрішній відступ, *border* - межа блоку, *margin* - зовнішній відступ, *width* - ширина області *content*, *height* - висота області *content*. Варто пам'ятати, що для CSS властивості `padding`, `margin`, `border` задаються за годинниковою стрілкою.

Досить часто важливо зробити так, щоб розрахунок блока відбувався від межі. У такому застосовують властивість `box-sizing`. Вона може набувати таких значень: `content-box` (при цьому значенні властивості width і height задають ширину і висоту контенту і не включають в себе значення відступів, полів та меж блоку), `border-box` (властивості width і height включають в себе значення полів і меж блоків, але не відступів margin), inherit (вказує на те що властивість буде успадковувати значення від свого батьківського елемента).

Важливою перевагою CSS є те, що завдяки йому дозволяється керувати (модифікувати) тип блоку. За це відповідає інструкція

`display` з такими основними значеннями `none, block, inline, inline-block`.

При побудові загальної структури важливо слідкувати за відображенням зовнішнього відступу браузером. Тут можуть виникнути дві наступні ситуації.

1. У випадку накладення двох вертикальних зовнішніх відступів `margin` браузер вибирає найбільший та встановлює його як основний. Додавання значень не відбувається.

2. При застосуванні `margin` до дочірніх елементів батьківського контейнера, властивість не буде застосовуватися до внутрішніх елементів, а натомість відступ буде зроблений для батьківського контейнера. Відбудеться «викидання» вертикальних відступів.

Окремо згадаємо про такі важливі технології як Flexbox і CSS Grid. Flexbox - це CSS-модуль, який визначає набір властивостей для розміщення, вирівнювання і розподілу простору між елементами у визначеному контейнері. Flex-контейнер - це батьківський групуючий елемент, завдяки налаштуванню якого можна керувати розміщенням дочірніх елементів. Типово до такого контейнера застосовуються такі властивості: `display, flex-direction, justify-content, align-items, flex-wrap, align-content.`

Flex-елементи - це теги, що поміщаються у flex-контейнері. До них застосовуються такі інструкції CSS: `flex-basis, flex-grow, flex-shrink, align-self, order.`

При роботі з технологією Flexbox варто пам’ятати про такі зауваження.

Елементи втрачають «тип», перестають бути рядковими або блоковими.

Блокові елементи перестають розташовуватись один під одним.

Крайні відступи на стику з межею батьківського не випадають.

Вертикальні відступи елементів не об’єднуються.

CSS Grid є прикладом технології, яка дозволяє задати набір властивостей для розміщення, вирівнювання і розподілу простору між елементами в контейнері.

Створення grid-контейнеру здійснюється шляхом розміщення у необхідному CSS класі, який прив'язаних до певного тегу HTML, властивість `display` із значеннями `grid`.

По суті Grid є перехресним набором горизонтальних і вертикальних ліній, що перетинаються. Один набір визначає стовпці, інший рядки. Елементи можуть бути розміщені на сітці, дотримуючись цих стовпців та рядків. Вона дозволяє забезпечити ряд переваг.

1. Позиціонування елементів за допомогою колонок та рядків як фіксованого розміру так і динамічно змінюваної ширини чи висоти у залежності від особливостей контенту.

2. Дозволяє керувати контентом, що перекривається.

Завдання до лабораторної роботи

Практичне завдання

Зверстати відповідно до варіанту головну сторінку сайт. Вона повинна відповідати наступним вимогам.

1. Обов'язковим є використання flexbox.
2. Структура документу має повністю відповідати макету, представленому у psd файлі.
3. Елементи меню мають змінюватися при наведенні на нього кнопки миші.
4. Колір шрифту, його розмір має наближатися до оригіналу.
5. Шрифт має підключатися автоматично до html файлу.
6. Забороняється використовувати таблиці.
7. Всі фали мають бути у репозиторії на сайті github.com або gitlab.com із назвою **Прізвище_NNg_Lab4 або** бути надіслано на електронну пошту викладача у **ZIP архіві**.
8. Посилання на репозиторій слід надіслати на пошту викладача.

Варіанти

Увага! Для перегляду варіантів рекомендується використовувати додаток Photopea URL: www.photopea.com. Для його використання знадобиться дозвіл браузера показувати спливаючі вікна.

1. Варіант URL: Lab4_V1.psd
2. Варіант URL: Lab4_V2.psd
3. Варіант URL: Lab4_V3.psd
4. Варіант URL: Lab4_V4.psd
5. Варіант URL: Lab4_V5.psd
6. Варіант URL: Lab4_V6.psd

Питання для самоперевірки

1. Що таке Flexbox і які його основні переваги у верстці web сторінок?
2. Які основні властивості контейнера Flexbox і як вони впливають на розташування елементів?

3. Що собою представляє технологія CSS Grid і як вона відрізняється від Flexbox?

4. Що таке властивість `flex-direction` і які значення вона може приймати?

5. Як використовуються властивості `grid-template-columns` та `grid-template-rows` для створення макета сітки?’

6. Як за допомогою властивості `grid-template-areas` можна створювати складні макети іменованих областей у CSS Grid?

Довідкові матеріали

1. Photopea URL: www.photopea.com
2. FlexBox URL: https://slides.com/mos_xandr/deck-4-7-11-25?authuser=0
3. Одиниці вимірювання URL: https://slides.com/mos_xandr/deck-4-7-11-54?authuser=0
4. CSS Grid URL: https://slides.com/mos_xandr/deck-6-9-12-58?authuser=0
5. CSS Flexbox URL: https://www.w3schools.com/css/css3_flexbox.asp\
6. CSS Grid Layout Module URL: https://w3schoolsua.github.io/css/css_grid_en.html#gsc.tab=0

ЛАБОРАТОРНА РОБОТА № 5. Поняття семантичної верстки.

Теоретичні відомості

Семантична верстка

Семантична верстка, як уже вище зазначалося, означає використання спеціальних тегів для надання вебсторінці чіткої структурованості матеріалу. Цей підхід дозволяє не тільки покращити сприйнятливість контенту для користувачів, але і полегшує роботу пошукових систем, а також технологій доступності, таких як екранні читачі. Семантичні теги, такі як `<header>`, `<footer>`, `<article>`, `<section>`, `<aside>` і `<nav>`, допомагають визначити різні частини сторінки та їхню функціональність. Наприклад, `<header>` використовується для позначення головної частини документа чи секції, тоді як `<footer>` визначає «підвал» сайту.

Однією з ключових переваг такого підходу до побудови вебсторінки є покращення SEO (пошукової оптимізації). Пошукові системи, такі як Google, використовують семантичні теги для більш точного розуміння структури та змісту, що може призвести до кращого індексування та підвищення рейтингу. Правильно структурований контент допомагає пошуковим системам визначити важливість і взаємозв’язок між різними частинами сторінки, що, у свою чергу, підвищує видимість й доступність контенту для користувачів.

Семантична верстка також сприяє доступності вебсторінок для людей з обмеженими можливостями. Екранні читачі та інші допоміжні технології можуть більш ефективно взаємодіяти з семантично правильно структурованим контентом, який дозволяє користувачам з вадами зору чи іншими обмеженнями краще розуміти структуру сайту та виконувати навігацію по ньому.

Варто також зауважити і на тому, що семантична верстка сприяє кращій підтримці та масштабованості коду. Веб-розробники, які працюють над проектами, можуть легше розуміти структуру документа завдяки логічно розташованим іменам тегів. Це знижує ймовірність помилок і спрощує процес оновлення й доповнення новими матеріалами й сервісами для веб-сайту.

Нарешті, семантична верстка сприяє покращенню користувацького досвіду. Користувачі можуть інтуїтивно розуміти, як виконувати навігацію по сторінці завдяки чітко визначеним секціям і елементам. Наприклад, `<nav>` тег чітко вказує на головне навігаційне меню, а `<article>` визначає окремі статті чи блоки контенту.

Завдання до лабораторної роботи

Зверстати відповідно до варіанту головну сторінку сайт. Вона повинна відповідати наступним вимогам.

1. Максимально зберегти відповідність між дизайном і сайтом.
2. Обов'язковим є дотримання семантичної структури html документу.
3. Ієрархія заголовків має бути максимально корректно оптимізована.
4. Елементи форми, якщо вони присутні у зразку, мають бути сформовані за допомогою відповідних тегів (наприклад забороняється посилання представляти кнопками і навпаки).
5. У заголовку першого рівня має бути представлений текст: "Виконав студент групи ___ фізико-математичного факультету ПІБ, варіант ___"
6. Заголовок першого рівня має бути скритим за допомогою CSS.
7. Забороняється використовувати таблиці.
8. Всі фали мають бути у репозиторії на сайті github.com або gitlab.com із назвою **Прізвище_NNg_Lab5** **або** бути надіслано на електронну пошту викладача у **ZIP архіві**.
9. Всі сторінки мають бути адаптивними.
10. Виконайте їх верстку.

Варіанти

Увага! Для перегляду варіантів рекомендується використовувати додаток Photopea URL: www.photopea.com. Для його використання знадобиться дозвіл браузера показувати спливаючі вікна.

1. Варіант URL: Lab_5b_V1.psd
2. Варіант URL: Lab_5b_V2.psd
3. Варіант URL: Lab_5b_V3.psd
4. Варіант URL: Lab_5b_V4.psd
5. Варіант URL: Lab_5b_V5.psd
6. Варіант URL: Lab_5b_V6.psd

Питання для самоперевірки

1. Що таке семантична верстка і які її основні цілі?
2. Які переваги надає використання семантичних тегів у HTML5 порівняно з несемантичними тегами, такими як `<div>` та `<span>`?
3. Які теги HTML5 вважаються семантичними і як вони використовуються?
4. Як семантична верстка покращує SEO (пошукову оптимізацію) веб-сторінок?

5. Чому семантична верстка важлива для доступності веб-сайтів для користувачів з обмеженими можливостями?

6. Що таке тег `<article>` і в яких випадках його слід використовувати?

7. Яка роль тега `<section>` у структурі HTML-документа і чим він відрізняється від `<div>`?

8. Як використання тегів `<header>`, `<footer>`, `<nav>`, `<aside>` впливає на структуру та розуміння веб-сторінки?

Довідкові матеріали

1. programming.in.ua Семантична верстка HTML5 URL: https://programming.in.ua/web-design/html/173-semantic-layout-html5

2. HTML: Semantic elements URL: https://code-basics.com/languages/html/lessons/semantic-elements

3. Fonts URL: https://docs.google.com/presentation/d/1tB_kGgY3XlXSflmToBthDzDx7NA8---i3zTRgEdAFA4/edit#slide=id.p

4. Position, Z-index URL: https://docs.google.com/presentation/d/1ai4JTc41Cszfp0V2mifJlSXjJsvE9rZlTdgqplFX_f8/edit#slide=id.p

5. Photopea URL: www.photopea.com

ЛАБОРАТОРНА РОБОТА № 6. Базові конструкції мови програмування JavaScript.

Теоретичні відомості

Мова програмування JavaScript

JavaScript (JS) — динамічна, об'єктно-орієнтована мова програмування і її основний напрям використання це створення web застосунків. Для того щоб вбудувати у HTML документ код, написаний за допомогою цієї мови використовують два підходи.

Застосування тега <script></script> між якими розміщується код програми.

Зовнішнє підключення файлу програми за допомогою атрибута `src` тега `<script></script>`. Файл скрипта має розширення `js` (приклад 6.1.).

Приклад 6.1.

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

Розглянемо докладніше особливості запису інструкцій. Для JS команди або синтаксичні конструкції рекомендується завершувати крапкою з комою (тут треба зауважити, що для останній версій мови програмування JavaScript це вже не є критичною вимогою). Коментарем вважається рядок, який починається з символу `//`, а багаторядковий коментар записується наступним чином `/*...*/`.

Для оголошення змінних використовуються два ключових слова `let` та `const`. За допомогою `let` визначається змінна блочного типу видимості і значення такої змінної може бути змінено у процесі виконання програми. У той же час інструкція `const` є оголошенням константи. Її значення можна лише зчитати, але не змінити.

Раніше використовувалося команда `var`, але зараз вона не застосовується по причині підвищення безпеки роботи з даними. Приклад 6.1 ілюструє особливості оголошення змінних.

Приклад 6.1.

```
let a = 12;
const = 15;
var b = 45; // Не використовується
```

При іменуванні змінних необхідно дотримуватися ряду правил.

1. Ім’я має містити лише букви, цифри або символи $ і _.

2. Перший символ не має бути числом.

3. Забороняється використовувати зарезервовані конструкції.

Окремо треба зауважити, що JavaScript є мовою програмування з динамічною типізацією. Це означає, що сама змінна не містить інформацію

про те, які дані зберігаються у ній, а є, по суті, посиланням на ці дані, а отже упродовж виконання програми одна змінна на різних етапах може посилатися на дані числового типу, рядок або ж «зберігати» об'єкт.

Примітивні типи даних

Серед базових примітивних типів даних варто виокремити такі.

`Boolean` – логічний тип даних

`Number` – цілі числа і дійсні числа. Якщо запис дуже великого цілого числа завершується маленькою літерою ***n***, то в такому випадку тип числа визначається як ***bigint***

`String` – рядки символів

`Symbol` – символьний тип даних

Також важливо вказати і на спеціальні типи даних, які також досить часто використовуються у процесі програмування.

`undefined` - змінна, якій не надали значення (неініціалізована змінна)

`null` - означає, що змінній надано умовно "порожнє значення"

Щоб визначити тип даних, з яким пов'язана певна змінна, використовують функцію `typeof(значення або змінна)` дозволяє повернути тип змінної чи даних.

Досить часто виникає ситуація, коли виникає необхідність трансформувати число, записане рядком, у звичне числове значення. Для цього застосовують функцію `Number(string)`, де `string` – рядкове значення. У той же час для перевірки чи є значення числом, використовують метод `isNaN` (приклад 6.2).

Приклад 6.2.

```
const value = "51";
console.log(Number.isNaN(value));// False
```

У окремих випадках з вказаного рядка символів необхідно виокремити числове значення. Тоді застосовується одна з наведених у прикладі 6.3 функцій.

Приклад 6.3.

```
console.log(Number.parseInt("12abs"));
// -> 12 - число
console.log(Number.parseFloat("12.67abs"));
// -> 12.67 - число
```

При перетворенні даних застосовують функції представлені у прикладі 6.4.

Приклад 6.4.

```
const value = String(123)
// Перетворення у рядок
const value = Number("1234")
// Перетворення у число
const value = Boolean("123")
// Перетворення у логічний тип (порожній рядок, null,
// 0 дає false,  все інше - true)
```

Оголошення рядків

Рядком у мові програмування JavaScript упорядкованою послідовністю символів, індексація яких розпочинається з 0 (приклад 6.5).

Приклад 6.5.

```
let st = "Hello!!!";
let st2 = 'Одинарні лапки';
let phrase = `Так можна вбудовувати у рядок значення
${st}`;
```

Останній рядок прикладу показує вбудовувати як можна виконувати вставлення виразів у завчасно сформований рядок.

Щоб отримати доступ до окремого символу рядка достатньо вказати назву змінної та квадратних дужках індекс того символу, який треба виокремити.

Типово, як і у всіх мовах високого рівня, рядки у JavaScript можна об’єднувати. Така операція називається конкатинація і позначається дією знаку додавання. Проте тут є певні особливості, особливо коли у записі присутні наприклад числа (приклад 6.6).

Приклад 6.6.

```
let result = 5 + "2";
console.log(result); // "52"
console.log(typeof (result)); // -> string
```

Як видно з прикладу результатом виконання дій буде отримано новий рядок і жодної помилки не буде виведено. Це можливо тому, що при виконанні

першого рядка число `5` буде переведено до рядкового типу і лише потім виконано дію конкатинації.

Складний тип даних - object

Розглянемо приклад 6.7.

Приклад 6.7.

```
let obj = {
  city: "Zhytomyr",
  age: 1040,
};
```

Він описує створення об'єкта у розглядуваній мові програмування. Об'єкт – це структура даних, яка дозволяє зберігати пари «ключ-значення». Тут:

`city` і `age` – це ключі (іноді їх називають властивостями або полями).

`"Zhytomyr"` і `1040` – це значення, що відповідають цим ключам.

Ключі використовуються для доступу до відповідних значень (приклад 6.8).

Приклад 6.8.

```
console.log(obj.city); // -> Zhytomyr
console.log(obj.age);  // -> 1040
```

Також дозволяється додавати нові або ж змінювати існуючі (приклад 6.9).

Приклад 6.9.

```
obj.country = "Ukraine";
obj.age = 1041;
console.log(obj);
// -> { city: 'Zhytomyr', age: 1041,
//country: 'Ukraine' }
```

Окрім додавання існує можливість видалення властивостей (приклад 6.10).

Приклад 6.10.

```
let obj = {
  city: "Zhytomyr",
  age: 1040,
};

obj.location = "Zhytomyr region"
console.log(obj.location)
delete obj.location
console.log(obj.location) // -> Error
```

Окремо варто загадати про спеціалізовані властивості. Для їх іменування можуть використовуватися як числові так і рядкові значення (приклад 6.11).

Приклад 6.11.

```
let obj = {
  city: "Zhytomyr",
  age: 1040,
  "Область": "Zhytomyr region",
};

console.log(obj["Область"]);
```

Щоб переглянуту всі властивості, що описані у такому типі даних як `object` варто застосовувати цикл перебору (приклад 6.12).

Приклад 6.12.

```
let obj = {
  city: "Zhytomyr",
  age: 1040,
};

for (let key in obj){
  console.log(obj[key]);
}
```

Команди введення /виведення даних

Досить часто для введення (виведення) даних застосовують модальні вікна. Ось ряд команд, які дозволяють оперувати ними для передачі або отримання інформації.

`alert("message")` – виводить повідомлення у модальне вікно та чекає доти, доки користувач не натисне кнопку “ОК”.

`console.log("message")` – виводить повідомлення у консоль браузера.

`prompt("message")` – виводить повідомлення та чекає доти, доки користувач не введе дані.

`confirm("message")` – показує повідомлення і чекає, коли користувач натисне “ОК” або “Скасувати”. Повертає `true` для ОК та `false` для “Скасувати”/Esc.

Основні оператори мови програмування JavaScript

Перш ніж розглянути типові оператори охарактеризуємо поняття операнду та оператора. Отже *операнд* – це значення (певні дані), до якого застосовується вказані у програмі оператори. Тоді *оператором* є спеціальний символ або набір символів, що визначає певну дію над одним або більше операндами (значеннями або змінними). Переважно вони можуть бути унарними (застосовується тільки до одного операнда) чи бінанрними (застосовується до 2-х операндів).

Розглянемо кожен з типів операторів більш детально. Найчастіше у програмуванні використовуються оператори арифметичних дій (приклад 6.13), порівняння (приклад 6.14) та дії, що дозволяють записати складні умови (приклад 6.15).

Приклад 6.13.

```
console.log(2 + 3) // додавання
console.log(2 - 7) // віднімання
console.log(2 * 7) // множення
console.log(2 / 7) // ділення
console.log(2 % 7) // остача від ділення
console.log(2 ** 7) // піднесення до степеня
```

Приклад 6.14.

```
>, < - більше / менше
>=, <= - більше / менше або дорівнює
```

```
== - дорівнює
!= - не дорівнює
=== - дорівнює і відповідні типи даних
!== - не дорівнює і відповідні типи даних
```

Приклад 6.15.

```
Логічне "і" - &&
Логічне "або" - ||
Логічне заперечення - !
```

Також варто згадати про оператори інкременту (**++**) та декременту(**--**), які відповідно збільшують (зменшують) значення змінної на 1.

Область видимості

Поняття «область видимості» змінних (variable scope) вказує, в яких частинах коду програми можна звернутися до певної змінної. Розрізняють *глобальну* та *локальну* видимість.

Глобальна область видимості використовується за замовчуванням. По суті це будь-яка змінна оголошена поза межами будь-якого блоку коду, що обмежується фігурними дужками. Вона доступна у будь-якій частині коду після її оголошення. Для створення глобальних змінних найчастіше використовують такі інструкції як `let` та `const`. Такий підхід до оголошення несе ряд вразливостей, оскільки їх завжди можна змінити у будь-якому місці коду.

Будь-яка конструкція, яка використовує фігурні дужки **{}** (оператор умови, цикли, функції та ін.) створює нову локальну область видимості, і змінні, оголошені в цій області видимості не будуть доступні поза цим блоком.

Також варто пам'ятати про принцип вкладеності областей видимості. Він полягає у тому, що кожна внутрішня область видимості (scope) має доступ до змінних зовнішньої області видимості, але зовнішня область не має доступу до змінних внутрішньої області. Це правило часто називають лексичним ланцюгом областей видимості (lexical scope chain).

Він працює наступним чином.

1. *Лексичний контекст*. У момент оголошення функції чи блоку JavaScript визначає, в якій області видимості знаходиться цей код.

2. *Ланцюг областей видимості*. Коли виконується код, JavaScript шукає змінні спочатку в поточній області видимості. Якщо змінна не знайдена,

пошук продовжується в зовнішній області, і так далі, поки не досягне глобальної області.

Оператор умови if / else

Оператор умови є основних, які дозволяють керувати потоком виконання коду. Його загальна конструкція представлена у прикладі 6.16.

Приклад 6.16.

```
if (умова){
  інструкція 1;
  …
  інструкція n;
}
else{
  інструкція 1;
  …
  інструкція n;
}
```

Його робота цілком аналогічна до решти його аналогів, що представлені в інших мовах програмування. Спочатку перевіряється умова, яка записана після ключового слова `if` , якщо вона істинна, то запускаються інструкції, що записані у фігурних дужках одразу ж після команди `if`. У випадку хибної умови виконується код, розміщений уже після – `else`.

Існують розширений варіант і версія без команд `else if` та `else` цієї конструкції (приклад 6.17, 6.18).

Приклад 6.17.

```
if (умова#1){
  набір інструкцій 1;
}
else if (умова#2){
  набір інструкцій 2;
}
else{
  набір інструкцій;
}
```

У випадку розширеного варіанта додається команда `else if`, а перевірка умов відбувається поступово доти, доки не буде виконана одна з них. Якщо такої ситуації не трапиться, виконується набір інструкції, які записані після ключового слова `else`.

Приклад 6.18.

```
if (умова){
  інструкція 1;
  …
  інструкція n;
```

Оскільки команди `else if` та `else` не є обов'язковими, то оператор умови можна представити з одним `if` та набором команд, які записуються після першої умови. У випадку не виконання цієї умови програма починає виконувати код, який написаний вже після конструкції оператора умови.

Скорочена форма оператора умови називається тернарним оператором (приклад 6.19). Досить часто його використовують для запису простої умови, що дозволяє обрати між двома варіантами значень або ж дій.

Приклад 6.19.

```
умова ? інструкція#1 : інструкція#2;
```

`Умова` – це вираз, який перевіряється, `інструкція#1` – виконується, коли умова істинна, а якщо ні то застосовується `інструкція#2`.

Досить часто тернарний оператор використовують для присвоєння значень змінної, за певних умов.

Оператор вибору switch / case

Оператор вибору є певною модифікацію оператора умови `if / else if / else` Його синтаксис представлено у прикладі 6.20.

Приклад 6.20.

```
let value = 10
switch (value){
  case 0:
    console.log("No");
    break;
  case 10:
```

```
    console.log("Yes");
    break;
  default:
    console.log("So-So");
}
```

Під час його виконання значення `value` порівнюється з кожним, яке записане після ключового слова `case`. Якщо відбувається співпадіння, то запускаються на команди, подані після двокрапки відповідного `case`. Для кожного випадку в кінці блоку команд має стояти команда `break`. Вона зупиняє виконання оператора вибору.

У випадку відсутності співпадінь виконується код після інструкції `default`.

Цикли

В мові програмування JavaScript виділяють три види циклів: цикл з передумовою, цикл з післяумовою та цикл перебору. Загалом циклічна конструкція у мовах програмування дозволяє виконувати спеціально записаний код визначену кількість разів. Розглянемо їх детальніше.

Цикл з передумовою (приклад 6.21) передбачає виконання тіла цикла доти доки умова записана після ключового слова `while` істинна.

Приклад 6.21.

```
let iter = 0;
while (iter < 10){
  console.log("Iter = ", iter);
  iter++;
}
```

На відміну від циклу з передумовою, цикл з післяумовою (приклад 6.22) завжди виконає свій набір інструкцій хоча б один раз. Після чого буде перевірена умова та прийнято рішення на повторення дій або на продовження роботи програми загалом.

Приклад 6.22.

```
let iter = 0;
do {
  console.log("Iter = ", iter);
```

```
  iter++;
}
while (iter < 10);
```

Цикл перебору (приклад 6.23) використовується для багаторазового виконання блоку коду з контрольованими параметрами.

Приклад 6.23.

```
for (let iter = 0; iter < 10; iter++){
  console.log("Iter = ", iter);
}
```

Зупинимося детальніше на записі у дужках. `let iter = 0` – початкове оголошення або присвоєння значення змінної, що використовується в циклі. `iter < 10` - вираз, який перевіряється перед кожною ітерацією. Якщо повертається `false`, цикл завершується. `iter++` - вираз, що виконується після кожної ітерації, наприклад, збільшення змінної.

При роботі з циклами важливо розуміти призначення таких команд як `continue` та `break`. Так `break` – передчасно завершує виконання циклу та виходить з нього, а `continue` – навпаки, передчасно завершує поточну ітерацію циклу та розпочинає наступну.

Завдання до лабораторної роботи

Розв’язати задачі подані у варіантах.

Обов’язковою є перевірка коректності введення даних та існування розв’язку.

Для кожної задачі має бути створена окрема html-сторінка, до якої додається файл з js скриптом.

Результати мають відображатися у консолі Веб-браузера форматі:

Задача 1

Текст задачі

Введені дані: ….

Результат виконання скрипта.

Всю роботу необхідно зберігати у репозиторії **Прізвище_NNg_JS_Lab_NN**.

Варіант 1

1. Записати вказане трицифрове натуральне число в зворотному порядку.
2. Перевірити, чи три вектора a, b, c, які задаються трьома координатами, є компланарними.
3. Знайти кількість чотиризначних чисел, сума цифр яких дорівнює k.
4. Розкласти натуральне число на прості множники.

Варіант 2

1. Записати дане чотирицифрове натуральне число без середніх цифр, які замінені підкресленням.
2. Ввести координати точок $A(x_0, y_0)$ і $B(x_1, y_1)$ і визначити яка з цих точок – A чи B – є найбільш віддалена від початку координат (O(0,0)). Відповідь вивести у вигляді повідомлення.
3. Знайти суму трицифрових чисел, які мають в своєму записі цифру d.
4. Знайти число паліндром, квадрат якого є також паліндром.

Варіант 3

1. Знайти площу бічної поверхні правильної чотирикутної піраміди об’ємом V і висотою h. результат вивести з точністю до тисячних.
2. Знайти найбільшу висоту трикутника із вказаними сторонами.
3. Задано деяке число N. Скласти програму пошуку суми простих чисел менших за N.
4. Із даного натурального числа вилучити всі нулі та замінити їх зірочками.

Варіант 4

1. Знайти остачу від ділення першої цифри на останню у заданому 5-ти цифровому натуральному числі.
2. На площині задано чотирикутник координатами своїх вершин. Визначити найбільшу з його сторін (її довжину).
3. Скласти програму розкладу числа на прості множники.
4. Знайти НСК двох двоцифрових натуральних чисел.

Варіант 5

1. Ввести координати точки A(x та y) і визначити: в якій чверті лежить ця точка. Відповідь вивести у вигляді повідомлення.
2. Ввести послідовність натуральних чисел та обчислити кількість двозначних чисел.

3. Вивести всі двозначні числа, які діляться на 9 або містять цифру 9.

4. Вивести НСК для довільної серії двоцифрових чисел від 20 до 40.

Варіант 6

1. Ввести координати точок $A_1(x_1,y_1)$, $A_2(x_2,y_2)$, $A_3(x_3,y_3)$ і визначити, чи лежать ці точки на одній прямій. Відповідь вивести у вигляді повідомлення у консолі.

2. Ввести послідовність натуральних чисел менших за N та обчислити кількість і суму тих членів послідовності, які діляться на 5 і не діляться на 7.

3. Вивести всі двозначні числа, сума квадратів цифр яких ділиться на 13.

4. Скільки цифр в даному числі кратні N? Чи правильно, що в даному числі сума першої та останньої дорівнює K?

Питання для самоперевірки

1. Як відбувається оголошення змінних в JS?.

2. Які операції можна виконувати над змінними і як визначається пріоритет виконання дій у JavaScript?

3. Як працює умовний оператор `if` у JavaScript, і які є його особливості?

4. Що таке оператор вибору `switch`, і коли доцільно його використовувати?

5. Які види циклів підтримує JavaScript, і як їх правильно використовувати?

6. Як у JavaScript реалізується введення та виведення даних?

Довідкові матеріали

1. Змінні JS URL: https://uk.javascript.info/variables

2. JavaScript Арифметичні оператори URL: https://w3schoolsua.github.io/js/js_arithmetic.html

3. Базові конструкції URL: https://docs.google.com/presentation/d/19Ftb8QzOEOyTQTM8tWCMWDBmyKQvx73f4ZE2eelg4SI/edit#slide=id.p

4. JavaScript Типи даних URL: https://w3schoolsua.github.io/js/js_datatypes.html

5. Умовні розгалуження: if, '?' URL: https://uk.javascript.info/ifelse

6. Конструкція "switch" URL: https://uk.javascript.info/switch

ЛАБОРАТОРНА РОБОТА № 7. Масиви і рядки в Java Script. Оголошення функцій.

Теоретичні відомості

Масиви

Масив у мові програмування JavaScript пердставляє собою сукупність елементів, які упорядковані за індексом. Кожному елементу масиву відповідає свій індекс (номер позиції). Нумерація індексів розпочинається з 0. Для того щоб створити масив використовують як літерал масиву так і його конструктор (приклад 7.1). Літерал представляє собою простий список розділених комами елементів у квадратних дужках.

Приклад 7.1.

```
let arr_empty =[];
let arr_fruit = ["Apple", "Orange", "Lemon" ];
let arr_nums = new Array(1, 2, 3, 4, 5);
```

Для того щоб отримати доступ до значення елемента масиву, необхідно вказати його індекс у квадратних дужках (приклад 7.2).

Приклад 7.2.

```
let arr_fruit = ["Apple", "Orange", "Lemon"];
console.log(arr[1]);// -> "Orange"
```

Розглядувана структура має можливість зберігати значення різних типів даних одночасно (приклад 7.3).

Приклад 7.3.

```
let arr_fruit = [40, "Orange", true];
console.log(typeof(arr_fruit[1]);// -> string
```

Щоб визначити кількість елементів, які розміщені у масиві, необхідно звернутися до властивості `length` (приклад 7.4).

Приклад 7.4.

```
let arr_fruit = [40, "Orange", true];
console.log(arr_fruit.length);// -> 3
```

Щоб помісти новий елемент у масив необхідно скористатися такими командами як `push(значення)` та `unshift(значення)` (приклад 7.5).

Приклад 7.5.

```
let arr_fruit = [40, "Orange"];
//Елемент додають у кінець масиву
arr_fruit.push(true);// -> [40, "Orange", true]
// Елемент додають на початок масиву
arr_fruit.unshift(true); // -> [true, 40, "Orange"]
```

За видалення елементів відповідають команди `pop()` та `shift()` (приклад 7.6).

Приклад 7.6.

```
let arr_fruit = [40, "Orange"];
//Елемент додають у кінець масиву
arr_fruit. pop();// -> [40]
// Елемент додають на початок масиву
arr_fruit.shift(); // -> ["Orange"]
```

Також можна додавати (видаляти) елементи в будь-якій частині масиву за допомого команди `splice` (приклад 7.7)

Приклад 7.7.

```
arr_fruit.splice(1, 1, "kiwi");
console.log(arr_fruit); // - >
// - > ["apple", "kiwi", "cherry"]
```

Щоб переглянути всі елементи в масиві використовують оператори циклів, найчастіше цикл перебору (приклади 7.8 – 7.10).

Приклад 7.8.

```
let arr = ["Apple", "Orange", "Lemon" ];
for (let i = 0; i < arr.length; i++){
  console.log(`Arr[${i}] = ${arr[i]}`);
}
```

Приклад 7.9.

```
let arr = ["Apple", "Orange", "Lemon" ];
```

```
for (let item of arr){
  console.log(item);
}
```

Приклад 7.10.

```
// Перебір ключів (індексів) масиву
let arr = ["Apple", "Orange", "Lemon"];
for (let key in arr){
  console.log(key);
}
```

Окремо зауважимо на багатовимірних масивах. Оскільки масив може містити елементи, які відносяться до різних типів, то така структура даних може містити як складові самі масиви. По суті багатовимірний масив це масив масивів (приклад 7.11).

Приклад 7.11.

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
console.log(matrix[1][2]);}
```

При роботі з масивами використовують досить багато інших методів. Назвемо деякі з них, що найчастіше використовуються. Зокрема `map()` – перетворює кожен елемент масиву; `filter()` – повертає елементи, що відповідають умові, котра записана у дужках; `forEach()` – виконує завчасно створену функцію для кожного елемента масиву; `sort()` – сортує елементи (за замовчуванням як рядки); `concat()` – об'єднує два або більше масивів тощо.

Функції та методи роботи з рядками

Вище було подано означення рядка та наведено приклади як їх можна задати. Тепер зосередимося на описі базових функцій та методів, які дозволяють швидко опрацьовувати такий тип даних.

Отже, для того щоб дізнатися кількість символів у певному заданому рядку варто скористатися властивістю `length` (приклад 7.12).

Приклад 7.12.

```
let str = "Hello";
console.log(str.length);// 5
```

За зміну регістра відповідають методи `toUpperCase()` та `toLowerCase()` (приклад 7.13).

Приклад 7.13.

```
console.log("hello".toUpperCase()); // "HELLO"
console.log("HELLO".toLowerCase()); // "hello"
```

Пошук у рядку можна виконати за допомогою функцій `indexOf(substring)`, `lastIndexOf(substring)`, `includes(substring)`, `startsWith(substring)` і `endsWith(substring)` (приклад 7.14).

Приклад 7.14.

```
Let str = "Hello, world!";
console.log(str.indexOf("world"));// 7
console.log(str.indexOf("planet"));// -1
console.log("abcabc".lastIndexOf("a")); // 3
console.log("hi world".includes("world")); // true
console.log("hello".startsWith("he")); // true
console.log("hello".endsWith("lo")); // true
```

Оперування з частинами (приклад 7.15) рядка здійснюється за допомогою методів `slice(start, end)` та `substring(start, end)`. Головна відмінність між ними у тому, що перший може працювати з від'ємними індексами, а другий ні.

Приклад 7.15.

```
console.log("hi, world!".slice(5, 10)); // "world"
console.log("hi, world!".substring(5, 10));
// -> "world"
```

За заміну символів (приклад 7.16) відповідають команди `replace(searchValue, newValue)` і `replaceAll(searchValue, newValue)` .

Приклад 7.16.

```
Let str = "I like JavaScript";
console.log(str.replace("JavaScript", "coding"));
// "I like coding"

let str = "I like JavaScript and JavaScript likes
me";
console.log(str.replaceAll("JavaScript", "coding");
// "I like coding and coding likes me"
```

Для вилучення пропусків варто застосовувати функцію `trim()` (приклад 7.17).

Приклад 7.17.

```
console.log("  Hello  ".trim());
// "Hello"
```

За розбиття та об'єднання кількох рядків в один відповідають функції й `split(delimiter)` та `join(separator)` (приклад 7.18).

Приклад 7.18.

```
let str = "apple,banana,cherry";
let fruits = str.split(","); console.log(fruits);
// ["apple", "banana", "cherry"]

let fruits = ["apple", "banana", "cherry"];
console.log(fruits.join(", "));
// "apple, banana, cherry");
```

Щоб вивести декілька разів певне повідомлення застосовують `repeat(count)` (приклад 7.18).

Приклад 7.19.

```
console.log("ha".repeat(3)); // "hahaha"
```

А лексикографічне порівняння рядків здійснюжться за допомогою метода `log` (приклад 7.20).

Приклад 7.20.

```
console.log("apple" > "banana"); // false
```

Звичайно це не повний перелік команд роботи з рядками, але перераховані функції цілком можна використовувати при вирішенні типових задач з обробки текстів та рядів у мові програмування JavaScript.

Робота з функціями у JS. Рекурсія.

Функції в JavaScript є фундаментальною частиною мови та використовуються для повторного використання коду. Їх можна створити кількома способами (приклад 7.21).

Приклад 7.21.

```
//Класичне оголошення
function greet(name){
    return `Hello, ${name}!`;
}
console.log(greet("John"));
// -> "Hello, John!"

//Функціональний вираз
const greet = function(name) {
    return `Hello, ${name}!`;};
console.log(greet("John"));
//  -> "Hello, John!"

//Стрілочна функція
const greet = (name) => `Hello, ${name}!`;
console.log(greet("John")); // "Hello, John!"

// Анонімні функці
setTimeout(function() {
    console.log("This is delayed!");
}, 1000);
```

Для JS функції є об’єктами типу Function, тому вони можуть бути: збережені у змінних, передані як аргументи, повернуті як результат іншої функції. Окрім цього також варто зауважити що кількість переданих

аргументів не обмежена, а до всіх переданих значень можна отримати доступ через об’єкт `arguments` (приклад 7.22).

Приклад 7.22.

```
function sum() {
    let total = 0;
    for (let arg of arguments) {
        total += arg;
    }
    return total;
}
console.log(sum(1, 2, 3, 4)); // 10
```

В окремих випадках важливо встановити значення для аргументів за замовчуванням. Такий варіант задання функції подано у прикладі 7.23.

Приклад 7.23.

```
function greet(name = "Guest") {
    return `Hello, ${name}!`;
}
console.log(greet());// "Hello, Guest!"
```

Для мови програмування JS характерним є використання замикання. Функція може «запам'ятовувати» доступ до змінних із зовнішнього блоку, навіть якщо цей блок уже завершив виконання (приклад 7.24).

Приклад 7.24.

```
function outer() {
    let count = 0;
    return function inner() {
        count++;
        return count;
    };
}
const counter = outer();
console.log(counter()); // 1
console.log(counter()); // 2
```

Також є можливість використовувати самовикликаючі функції (приклад 7.25).

Приклад 7.25.

```
(function() {
    console.log("IIFE executed!");
})();
```

Окремо зупинимося на понятті рекурсії (функція викликає сама себе). Вона дозволяє вирішувати задачі, які мають ітеративну природу, наприклад обхід деревоподібних структур (приклад 7.26).

Приклад 7.26.

```
const tree = {
    value: 1,
    children: [
        {
            value: 2,
            children: [
                { value: 4, children: [] },
                { value: 5, children: [] },
            ],
        },
        {
            value: 3,
            children: [
                { value: 6, children: [] },
                { value: 7, children: [] },
            ],
        },
    ],
};

function traverse(node) {
    console.log(node.value);
    for (let child of node.children) {
        traverse(child);
    }
}
```

```
traverse(tree);
```

Завдання до лабораторної роботи

Розв’язати задачі подані у варіантах.

Обов’язковою є перевірка коректності введення даних та існування розв’язку.

Для кожної задачі має бути створена окрема html-сторінка, до якої додається файл з js скриптом.

Результати мають відображатися у консолі Веб-браузера форматі:

Задача 1

Текст задачі

Введені дані: ….

Результат виконання скрипта.

Всю роботу необхідно зберігати у репозиторії **Прізвище_NNg_JS_Lab_NN**.

Варіант 1

1. В даному масиві із N цілих чисел знайти суму останніх 5 додатніх елементів.
2. Обчислити факторіал заданого числа рекурсивно.
3. Створіть функцію, яка розраховує довжину масиву і повертає її у вигляді числа. Масив в функцію передається як аргумент. Якщо аргумент не заданий - виводиться повідомлення про помилку (без використання властивості `length`).

Варіант 2

1. У автоматично згенерованому масиві цілих чисел, знайти число, яке зустрічається найчастіше. Якщо таких чисел декілька, то вивести їх всі та вказати їх кількість.
2. З клавіатури вводиться текстовий рядок. Скласти скрипт, який підраховує кількість слів у кожному реченні.
3. Обчисліть n-те число Фібоначчі.

Варіант 3

1. В даному масиві із n дійсних чисел знайти добуток перших 5 від’ємних елементів.

2. З клавіатури вводиться текстовий рядок. Скласти скрипт, який виводить всі символи після першого символу “:”.

3. Користувач вводить числа. Якщо число більше 10, то функція повертає квадрат числа, якщо менше 7 - пише, що число менше 7. Якщо 8, 9 - то повертає відповідно 7 або 8. Реалізуйте рішення з декількома return.

Варіант 4

1. З клавіатури вводиться текстовий рядок. Скласти скрипт, який замінює всі великі літери, на відповідні малі.

2. Обчислити степінь натурального числа рекурсивно.

3. Напишіть гру «Вгадай число». При завантаженні сторінки генерується випадкове число від 0 до 10. Користувачу надається три спроби вгадати число. При кожній перевірці видається підказка: більше чи менше. При вгадуванні або завершенні числа спроб видається оповіщення. Кількість спроб виводиться на екран. Результат для цієї задачі виводиться за допомогою команди `alert()`.

Варіант 5

1. У автоматично згенерованому масиві із 12 цілих чисел знайти всі, які кратні 5 та вивести їх у рядок відсортованим за зростанням.

2. З клавіатури вводиться текстовий рядок. Написати функцію, яка розраховує кількість чисел у рядку (числом вважається послідовність символів, яка починається із цифри 1 - 9, у своєму записі містить знаки 0 - 9 і крапку та відокремлена пропусками).

3. Функція `func` приймає 2 параметра: число і анонімну функцію, яка підносить число до квадрату. Піднесіть число до 4-тої степеня за допомогою `func`.

Варіант 6

1. Задано масив із 16 чисел, що генерується автоматично. Вивести у рядок всі елементи, які розміщені у парних позиціях, та відсортувати його у порядку спадання.

2. З клавіатури вводиться числове значення вартості. Записати текстове представлення вартості у гривнях. Максимальне значення, яке може бути введене у вигляді числа це 1000.

3. Зробіть функцію each, яка першим параметром приймає масив, а другим - функцію, яка застосовується до кожного елементу масиву. Функція each повинна повернути змінений масив.

Питання для самоперевірки

1. Як оголосити масив у JavaScript?
2. Які вбудовані функції існують для роботи з масивами у JavaScript?
3. Що таке рядки у JavaScript?
4. Які вбудовані функції використовуються для роботи з рядками у JavaScript?
5. Як оголосити функцію у JavaScript?
6. Що таке рекурсія і як вона працює у JavaScript?

Довідкові матеріали

1. Масиви URL: https://uk.javascript.info/array
2. Масиви. Функції.Контекст. URL: https://docs.google.com/presentation/d/1KR7ZHdNFCQg7xbsxoe2ty433aW5i8kegNDSrNFU7f_g/edit#slide=id.p
3. Методи масивів URL: https://uk.javascript.info/array-methods
4. Рядки URL: https://uk.javascript.info/string
5. Функції URL: https://uk.javascript.info/function-basics
6. Рекурсія та стек URL: https://uk.javascript.info/recursion

ЛАБОРАТОРНА РОБОТА № 8. Робота із DOM у JavaScript.

Теоретичні відомості

Поняття ООП в JS

Об’єктно-орієнтоване програмування (ООП) у JavaScript – це підхід до програмування, у якому програми моделюються як набір об’єктів, що взаємодіють між собою. Кожен об’єкт має свої властивості (дані) і методи (функції, які реалізують поведінку). Поняття об’єкту вже описувалося вище, тому більшою мірою зосередимося на визначенні класу.

Клас у JavaScript фактично (приклад 8.1) є певним шаблоном для створення об’єктів. Вони з’явилися в стандарті ES6 і дозволяють реалізувати ООП більш структуровано.

Приклад 8.1.

```
class Animal {
  constructor(name, sound) {
    this.name = name;
    this.sound = sound;
  }
  makeSound(){
    console.log(`${this.name} каже: ${this.sound}`);
  }
}

const dog = new Animal("Пес", "гав-гав");
dog.makeSound(); // Пес каже: гав-гав
```

Для будь-якої об’єктно-орієнтованої мови програмування характерним є інкапсуляція (об’єкти приховують свою внутрішню реалізацію від зовнішнього світу, надаючи доступ лише до певного інтерфейсу – набору методів); наслідування (один клас може успадковувати властивості та методи іншого класу); поліморфізм (дозволяє методам з однаковими назвами поводитися по-різному залежно від контексту). Приклади 8.2 – 8.4.

Приклад 8.2.

```
// інкапсуляція
class Account {
    #balance = 0; // Приватна властивість

    deposit(amount) {
        this.#balance += amount;
```

```
        }

        getBalance() {
            return this.#balance;
        }
    }

    const myAccount = new Account();
    myAccount.deposit(100);
    console.log(myAccount.getBalance()); // 100
```

Приклад 8.3.

```
    // наслідування
    class Vehicle {
        constructor(type, speed) {
            this.type = type;
            this.speed = speed;
        }
        move() {
            console.log(`${this.type} рухається зі швидкістю
${this.speed} км/год`);
        }
    }

    class Car extends Vehicle {
        constructor(brand, speed) {
            super("Автомобіль", speed);
    // Виклик конструктора батьківського класу
            this.brand = brand;
        }
        honk() {
            console.log(`${this.brand} сигналить!`);
        }
    }

    const myCar = new Car("Tesla", 120);
    myCar.move();
    // Автомобіль рухається зі швидкістю 120 км/год
    myCar.honk();
    // Tesla сигналить!
```

Приклад 8.4.

```
// поліморфізм
class Shape {
    draw() {
        console.log("Малюється форма");
    }
}

class Circle extends Shape {
    draw() {
        console.log("Малюється коло");
    }
}

class Square extends Shape {
    draw() {
        console.log("Малюється квадрат");
    }
}

const shapes = [new Shape(), new Circle(), new Square()];
shapes.forEach(shape => shape.draw());
// Виведе:
// Малюється форма
// Малюється коло
// Малюється квадрат
```

Особливості роботи з DOM

DOM (Document Object Model) – це об’єктна модель HTML- або XML-документа, яка представляє структуру сторінки у вигляді дерева вузлів. За допомогою DOM можна динамічно змінювати вміст, структуру та стилі веб-сторінки через JavaScript.

Для оперування елементами такої структури використовують наступні методи.

`document.getElementById(id)` – знаходить елемент за його `id`.

`document.getElementsByClassName(className)` – повертає всі елементи з певним класом.

`document.getElementsByTagName(tagName)` – знаходить елементи за тегом (наприклад, div, p).

`document.querySelector(selector)` – вибирає перший елемент за CSS-селектором.

`document.querySelectorAll(selector)` – повертає всі елементи, які відповідають селектору.

З усього наведеного переліку насьогодні найчастіше використовують `document.getElementById(id)`, `document.querySelector(selector)` та `document.querySelectorAll(selector)`.

Щоб змінити вміст та атрибути елементів використовують переважно такі властивості та методи вузлів DOM.

`innerText/textContent` – змінюють текст елемента.

`innerHTML` – змінює HTML-код усередині елемента.

`setAttribute()` `/getAttribute()` – додають або отримують значення атрибута.

`hasAttribute(name)` – перевіряє наявність.

`removeAttribute()` – видаляє атрибут.

Окремо варто згадати і модифікацію класів каскадних таблиць стилів. Так `element.style` – змінює інлайн-стилі, `classList` – дозволяє працювати безпосередньо з класами CSS. Тут широко використовуються дакі методи як: `add()` – додає клас, `remove()` – видаляє клас, `toggle()` – додає клас, якщо його немає, або видаляє, якщо є, `contains()` – перевіряє, чи є клас у елемента.

Додавання та видалення елементів дерева DOM передбачає ряд кроків. Створити сам елемент за допомогою команд `document.createElement(tag)` або `document.createTextNode(text)`. Далі, щоб він відобразився необхідно додати у саме дерево – `document.body.append(tag)`. Для останньої дій також можуть застосовуватися такі методи.

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

`node.prepend(...вузли або рядки)` – вставляє вузли вбо рядки на початку `node`,

`node.before(...вузли або рядки)` – вставляє вузли або рядки попереду `node`,

`node.after(...вузли або рядки)` – вставляє вузли або рядки після node,

`node.replaceWith(...вузли або рядки)` – замінює `node` заданими вузлами або рядками.

Також можливо виконати безпосередню вставку коду за допомогою команди `elem.insertAdjacentHTML(куди, html)`. Перший параметр це кодове слово, яке вказує куди вставляти відносно `elem`. Його значеннння має бути одним з наступних: «`beforebegin`» – вставити html безпосередньо перед `elem`; «`afterbegin`» – вставити html в `elem`, на початку; «`beforeend`» – вставити html в `elem`, в кінці, «`afterend`» – вставити html безпосередньо після `elem`. Другий параметр це рядок у форматі HTML.

Щоб видалити вузол варто використовувати метод `node.remove()`.

Окрім цього важливо розуміти як відбувається навігація по DOM-дереву. Зокрема застосовуються такі властивості елементів.

`parentNode` – батьківський вузол.

`childNodes` – список дочірніх вузлів.

`nextSibling` і `previousSibling` – сусідні вузли.

`closest(selector)` – найближчий предок, який відповідає селектору.

Ще однією важливим елементом роботи з DOM є опрацювання подій, які відбуваються на сторінці. Подія – це сигнал від браузера про те, що щось сталося. Вони використовуються для реакції на дії відвідувача або ж виконання коду. При опрацюванні подій, вони стають у чергу та обробляються у порядку надходження, асинхронно і незалежно одна від одної. Розрізняють такі основні види подій.

Події миші

`click` - відбувається, коли відбувся клік на елементі лівою кнопкою миші.

`focus` - відвідувач фокусується на елементі, наприклад, натискає на `input`.

`contextmenu` – відбувається, коли клікнули на елементі правою кнопкою миші.

`mouseover` / `mouseout` – коли миша наводиться на / залишає елемент.

`mousedown` / `mouseup` – коли натиснули / відпустили кнопку миші на елементі.

`mousemove` – під час руху миші.

Події клавіатури

`keydown` та `keyup` – коли користувач натискає / відпускає клавішу.

`keypress` – спрацьовує, коли натиснута функціональна клавіша.

Події елементів форми

`submit` – користувач надіслав форму `form`.

`focus` – користувач фокусується на елементі, наприклад, натискає на `input`.

`DOMContentLoaded` – коли HTML завантажено й оброблено, DOM документа повністю побудований і доступний.

`transitionend` – коли CSS-анімацію завершено.

Для того, щоб елемент реагував на дії користувача, до нього необхідно додати слухача (обробника) події й задати функцію, яка її буде опрацьовувати. Саме завдяки слухачам подій скрипт може реагувати на дії користувача.

Методи `elem.addEventListener()` і `elem.removeEventListener()` це спосіби, які дозволяють призначити або видалити відповідний обробник (слухач подій). Також варто зауважити, що при цьому можна використовувати довільну кількість обробників на одному типі події.

Опишемо детальніше перший – `element.addEventListener(event, handler, [options]);`

`event` – назва події, наприклад `click`.

`handler` – посилання на функцію, що виконуватиме опрацювання події.

`options` – додатковий об’єкт із властивостями.

`once` – якщо true, тоді обробник буде автоматично вилучений після виконання.

`capture` – фаза, на якій повинен спрацювати обробник.Так історично склалося, що `options` може бути `false/true`, це те саме, що `{capture: false/true}`.

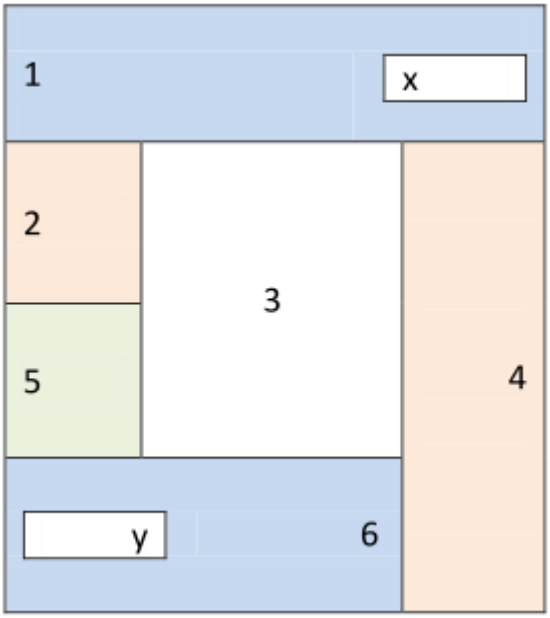
`passive` – якщо true, тоді обробник ніколи не викличе `preventDefault()`.

Завдання до лабораторної роботи

Розв’язати задачі подані у варіантах.

Всю роботу необхідно зберігати у репозиторії **Прізвище_NNg_JS_Lab_NN**.

Варіант 1



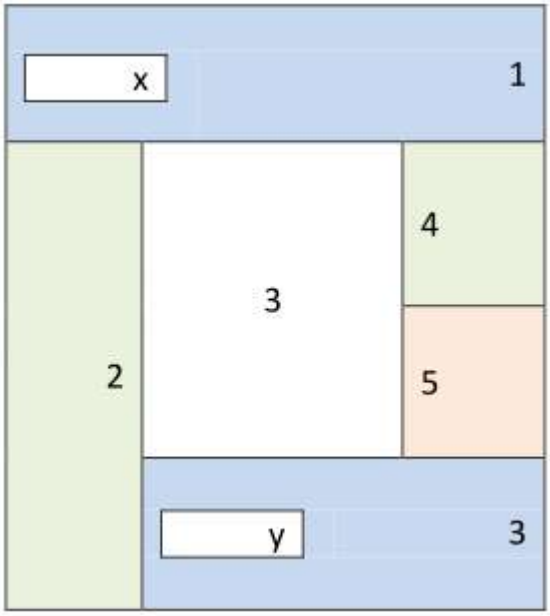
1. Зверстайте систему блоків, таку як показано на рисунку.

2. Поміняйте місцями контент блоків «x» та «y». Кнопка, яка виконує цю дію розміщена у полі 4 і має назву “Задача 2”.

3. Напишіть функцію, яка обчислює площу кола, беручи необхідні значення із спеціалізованих полів блоку 3, що генеруються автоматично при натисканні на відповідну кнопку, яка розміщена у блоці 4 та має назву “Задача 3”, і виводить отриманий результат в кінці контенту в блоці «3».

4. Напишіть скрипт, який визначає кількість максимальних чисел із 10 значень, беручи необхідні значення із відповідної форми (кнопка, яка створює таку форму розміщена у блоці 4 і має назву “Задача 4”) в блоці «3» на сторінці, а отриманий результат виводить за допомогою діалогового вікна на екран і зберігає в куках. Додаткові дані: а) при оновленні сторінки за допомогою діалогового вікна на екрані виводиться інформація, збережена в куках, із питанням про необхідність видалити дані із куків, і не виводиться згадана вище форма; б) при підтвердженні питання відповідні куки видаляються, і сторінка оновлюється з початковим станом із наявною формою для введення даних; в) при відмові виводиться наступне діалогове вікно із інформуванням користувача про наявність куків і необхідність перезавантаження сторінки.

Варіант 2



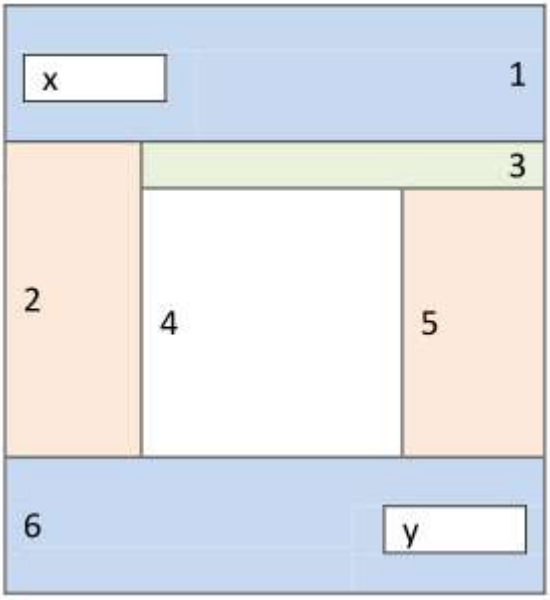
1. Зверстайте систему блоків, таку як показано на рисунку.

2. Поміняйте місцями контент блоків «4» та «5». Кнопка, яка виконує цю дію розміщена у полі 2 і має назву “Задача 2”.

3. Напишіть функцію, яка обчислює площу трикутника, беручи необхідні значення із спеціалізованих полів блоку 3, що генеруються автоматично при натисканні на відповідну кнопку, яка розміщена у блоці 2 та має назву “Задача 3”, і виводить отриманий результат в кінці контенту в блоці «3».

4. Напишіть скрипт, який визначає кількість мінімальних чисел із 10 значень, беручи необхідні значення із відповідної форми (кнопка, яка створює таку форму розміщена у блоці 2 і має назву “Задача 4”)в блоці «3» на сторінці, а отриманий результат виводить за допомогою діалогового вікна на екран і зберігає в куках. Додаткові дані: а) при оновленні сторінки за допомогою діалогового вікна на екрані виводиться інформація, збережена в куках, із інформуванням, що після натискання кнопки «ОК» відбудеться видалення даних із куків, і не виводиться згадана вище форма; б) при натисканні кнопки «ОК» відповідні куки видаляються, і виводиться наступне діалогове вікно із повідомленням, що куки видалено, а натискання кнопки «ОК» перезавантажує сторінку з початковим станом із наявною формою для введення даних.

Варіант 3



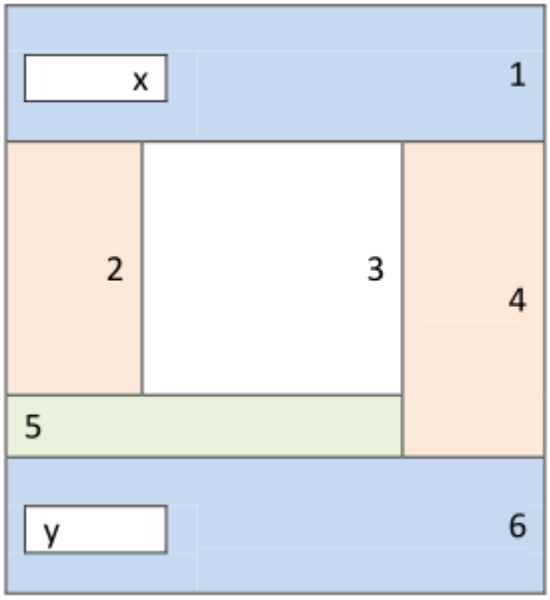
1. Зверстайте систему блоків, таку як показано на рисунку.

2. Поміняйте місцями контент блоків «x» та «y». Кнопка, яка виконує цю дію розміщена у полі 2 і має назву “Задача 2”.

3. Напишіть функцію, яка обчислює площу прямокутника, беручи необхідні значення із спеціалізованих полів блоку 4, що генеруються автоматично при натисканні на відповідну кнопку, яка розміщена у блоці 2 та має назву “Задача 3”, і виводить отриманий результат в кінці контенту в блоці «4».

4. Напишіть скрипт, який визначає мінімальне і максимальне числа із 10 значень, беручи необхідні значення із відповідної форми (кнопка, яка створює таку форму розміщена у блоці 2 і має назву “Задача 4”) в блоці «4» на сторінці, а отриманий результат виводить за допомогою діалогового вікна на кран і зберігає в куках. Додаткові дані: а) при оновленні сторінки за допомогою діалогового вікна на екрані виводиться інформація, збережена в куках, із питанням про необхідність зберегти дані із куків, і не виводиться згадана вище форма; б) при підтвердженні питання виводиться наступне діалогове вікно із інформуванням користувача про наявність куків і необхідність перезавантаження сторінки; в) при відмові відповідні куки видаляються, і сторінка оновлюється з початковим станом із наявною формою для введення даних.

Варіант 4



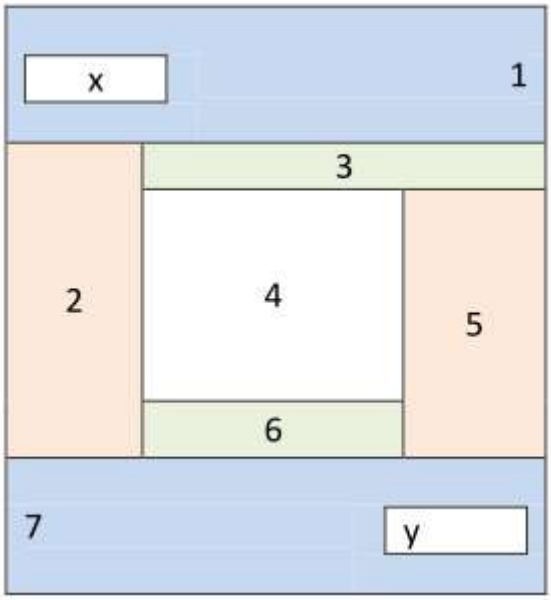
1. Зверстайте систему блоків, таку як показано на рисунку.

2. Поміняйте місцями тексти, позначені «1» та «6». Кнопка, яка виконує цю дію розміщена у полі 4 і має назву “Задача 2”.

3. Напишіть функцію, яка обчислює площу ромба, беручи необхідні значення із спеціалізованих полів блоку 3, що генеруються автоматично при натисканні на відповідну кнопку, яка розміщена у блоці 4 та має назву “Задача 3”, і виводить отриманий результат в кінці контенту в блоці «3».

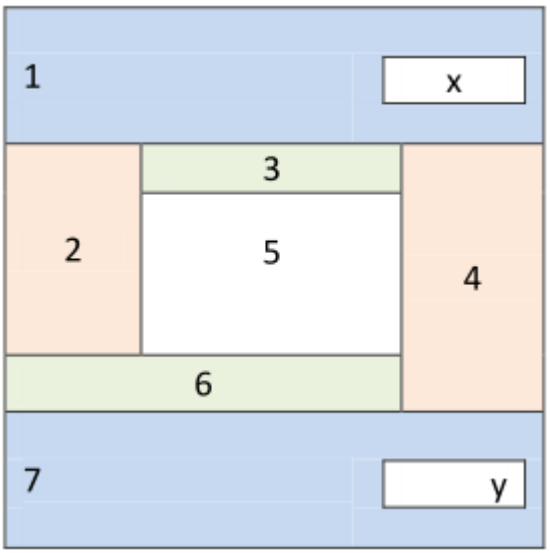
4. Напишіть скрипт, який визначає можливість побудови трикутника із заданими довжинами сторін, беручи необхідні значення із відповідної форми (кнопка, яка створює таку форму розміщена у блоці 4 і має назву “Задача 4”) в блоці «3» на сторінці, а отриманий результат виводить за допомогою діалогового вікна на екран і зберігає в куках. Додаткові дані: а) при оновленні сторінки за допомогою діалогового вікна на екрані виводиться інформація, збережена в куках, із питанням про необхідність видалити дані із куків, і не виводиться згадана вище форма; б) при підтвердженні питання відповідні куки видаляються, і сторінка оновлюється з початковим станом із наявною формою для введення даних; в) при відмові виводиться наступне діалогове вікно із інформуванням користувача про наявність куків і необхідність перезавантаження сторінки.

Варіант 5



1. Зверстайте систему блоків, таку як показано на рисунку.
2. Поміняйте місцями контент блоків «x» та «y». Кнопка, яка виконує цю дію розміщена у полі 5 і має назву “Задача 2”.
3. Напишіть функцію, яка обчислює площу п’ятикутника, беручи необхідні значення із спеціалізованих полів блоку 4, що генеруються автоматично при натисканні на відповідну кнопку, яка розміщена у блоці 5 та має назву “Задача 3”, і виводить отриманий результат в кінці контенту в блоці «4».
4. Напишіть скрипт, який «перевертає» задане натуральне число (125 -> 521), беручи необхідне значення із відповідної форми (кнопка, яка створює таку форму розміщена у блоці 5 і має назву “Задача 4”) в блоці «4» на сторінці, а отриманий результат виводить за допомогою діалогового вікна на екран і зберігає в куках. Додаткові дані: а) при оновленні сторінки за допомогою діалогового вікна на екрані виводиться інформація, збережена в куках, із інформуванням, що після натискання кнопки «ОК» відбудеться видалення даних із куків, і не виводиться згадана вище форма; б) при натисканні кнопки «ОК» відповідні куки видаляються, і виводиться наступне діалогове вікно із повідомленням, що куки видалено, а натискання кнопки «ОК» перезавантажує сторінку з початковим станом із наявною формою для введення даних.

Варіант 6



1. Зверстайте систему блоків, таку як показано на рисунку.

2. Поміняйте місцями контент блоків «3» та «6». Кнопка, яка виконує цю дію розміщена у полі 4 і має назву “Задача 2”.

3. Напишіть функцію, яка обчислює площу трапеції, беручи необхідні значення із спеціалізованих полів блоку 5, що генеруються автоматично при натисканні на відповідну кнопку, яка розміщена у блоці 4 та має назву “Задача 3”, і виводить отриманий результат в кінці контенту в блоці «5».

4. Напишіть скрипт, який визначає всі дільники заданого натурального числа, беручи це число із відповідної форми (кнопка, яка створює таку форму розміщена у блоці 4 і має назву “Задача 4”) в блоці «5» на сторінці, а отриманий результат виводить за допомогою діалогового вікна на кран і зберігає в куках. Додаткові дані: а) при оновленні сторінки за допомогою діалогового вікна на екрані виводиться інформація, збережена в куках, із питанням про необхідність зберегти дані із куків, і не виводиться згадана вище форма; б) при підтвердженні питання виводиться наступне діалогове вікно із інформуванням користувача про наявність куків і необхідність перезавантаження сторінки; в) при відмові відповідні куки видаляються, і сторінка оновлюється з початковим станом із наявною формою для введення даних.

Питання для самоперевірки

1. Що таке DOM, і яке його значення у JavaScript?
2. Які існують види вузлів DOM, і як вони використовуються?
3. Що являє собою об'єкт `document` у JavaScript?
4. Які основні властивості об'єкта `document` існують, і для чого вони використовуються?

5. Що таке браузерна подія, і як її можна перехоплювати у JavaScript?

6. Як працює порядок опрацювання подій у DOM? Що таке "захоплення" і «спливання»?

7. Що таке об’єкт події, і які дані він містить?

Довідкові матеріали

1. DOM дерево URL: https://uk.javascript.info/dom-nodes
2. Об'єктна модель документа HTML URL: http://tilsite.ho.ua/ur9-10.html
3. Вступ до подій браузера URL: https://uk.javascript.info/introduction-browser-events
4. Запуск користувацьких подій URL: https://uk.javascript.info/dispatch-events
5. JavaScript Об’єкти URL: https://w3schoolsua.github.io/js/js_objects.html
6. HTML DOM Events URL: https://www.w3schools.com/jsref/dom_obj_event.asp
7. Element: click event URL: https://developer.mozilla.org/en-US/docs/Web/API/Element/click_event
8. Події: change, input, cut, copy, paste URL: https://uk.javascript.info/events-change-input
9. CSS властивість animation URL: https://css.in.ua/css/property/animation
10. Як створити анімації CSS (з прикладами) URL: https://www.freecodecamp.org/ukrainian/news/yak-stvoryty-animatsiyu-css-z-prykladamy/

ДОДАТКИ

Завдання підвищеної складності для роботи з DOM #1[4]

Завдання 1

Створіть сторінку для конвертації валюти (долари США → біткойн). Сторінка має містити поля, в які вводиться нинішній курс біткоїна та кількість доларів СШ, які ви маєте конвертувати.

При заповненні або зміні данних сторінка повинна миттєво виводити результату вигляді блока (div) або параграф (тег <p>), який повідомляє скільки біткоінів ви можете купити.

Приклад інтерфейсу див. URL: Var1_1

Завдання 2

Зверстайте інтерфейс сторінки за прикладом. Під час натискання на кнопку «Видалити», новостворений блок зникає. Якщо відбувається натискання на кнопці «Редагувати» блок переходить в стан редагування (наприклад, з допомогою textarea). При знятті фокуса з textarea, блок повертається в звичайний стан.

Приклад інтерфейсу див. URL:Var1_2

Завдання 3

Створіть сторінку для конвертації валют (українська гривня → долари США), яка містить наступні поля:

1) Поле з курсом долара
2) Поле кількості грошей
3) Кнопка обрахувати
4) Кнопка очистити все
5) Поле з результатом

Приклад інтерфейсу див.:Var2_1

Завдання 4

Створіть сторінку містить два блока тексту. При натисканні на кнопку SWAP тексти блоків міняються місцями та для кожного обраховується кількість речень, слів та символів у текстах.

Приклад інтерфейсу див URL: Var2_2

Завдання 5

Створіть сторінку для розрахунку кінцевої суми депозиту в банку, при умові, що капіталізація відбувається щомісячно.

Приклад інтерфейсу див URL: Var3_1

Завдання 6

---

[4] Для пошуку зображень використовуйте https://www.svgviewer.dev

Створіть сторінку з присутніми коментарями. Реалізуйте можливість створити власний коментар та додайте модуль, який буде виводити кількість коментарів.

Приклад інтерфейсу див URL: Var3_2

Завдання 7

Створіть сторінку для розрахунку терміну виплат по споживчому кредиту. Відсотки не капіталізуються.

Приклад інтерфейсу див URL: Var4_1.png

Завдання 8

Створіть сторінку з коментарями та реалізуйте можливість ставити лайк/дизлайк з присутньою логікою перемикача та виведіть кількість позитивних поміток.

Приклад інтерфейсу див. URL: Var4_2.png

Завдання 9

Створіть сторінку, при завантаженні якої генерується випадаючий список з 5 основних світових валют та їх курсу до гривні.

Приклад інтерфейсу див. URL: Var5_1.png

Завдання 10

Створіть сторінку, яка дозволяє згенерувати поле у якому буде відображатися чат спілкування двох користувачів (користувач 1 та користувач 2).

Приклад інтерфейсу див URL: Var5_2.png

Завдання 11

Створити калькулятор для побудови стовпчастої діаграми (3 стовпчика мінімум).

Приклад інтерфейсу див. URL: Var6_1.png

Завдання 12

Створити сторінку, яка містить поле введення пароля, яке відображає ступінь його надійності.

Приклад інтерфейсу див. URL: Var6_2.png

Завдання підвищеної складності для роботи з DOM #2[5]

Завдання 1

Необхідно виконати такі умови завдання для різних типів інтерфейсів.

- Зверстати сторінку за зразком
- Реалізувати логіку додавання товарів до корзини за допомогою JavaScript.
- Перевіряти наявність товару перед додаванням: якщо кількість товару в корзині вже досягла наявної кількості, заборонити додавання нових одиниць.
- Розробити логіку для підрахунку кінцевої суми товарів у корзині на основі вибраних користувачем товарів.
- Перелік з доступними товарами має бути зверстано поряд з корзиною і зверстаний у тому ж стилі. Для доступного товару з переліку має бути вказана його ва одиницю. Також має міститися кнопка, яка додає товар у корзину.
- Створити логіку, яка дозволяє реалізувати таку умову: якщо кількість екземплярів певного товару рівна 0, то він має бути вилученим з корзини. Також має бути передбачено видалення товару за допомогою кнопки.

Приклад інтерфейсу № 1. Див. URL: Var1_img.png
Приклад інтерфейсу № 2. Див. URL: Var2_img.jpg
Приклад інтерфейсу № 3. Див. URL: Var3_img.jpg
Приклад інтерфейсу № 4. Див. URL: Var4_img.png
Приклад інтерфейсу № 5. Див. URL: Var5_img.png
Приклад інтерфейсу № 6. Див. URL: Var6_img.jpg

Завдання 2

Розробити вебзастосунок, який дозволяє користувачам вибирати кольори з палітри і заливати маленькі квадратні блоки на прямокутному полотні обраним кольором (рис. 2.1).

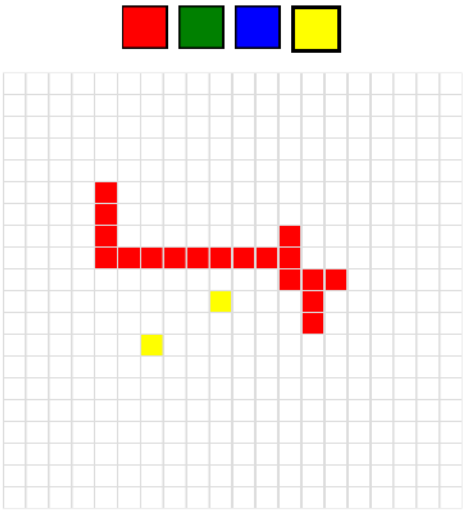


Рис. 2.1. Приклад інтерфейсу до завдання 2.

[5] Макети завданнь 1 - 6 можна знайти за посиланням https://drive.google.com/drive/folders/19zCnfQVOlbBFSL7wVDl0Qy61Ezv-uhPB?usp=sharing

Завдання 3

Є ряд гральних карт, повернутих до нас «сорочкою». При клацанні на карту, вона повертається до нас лицевою стороною протягом 1-2 секунд (повинен бути саме поворот). При повторному клацанні, карта знову повертається «сорочкою» (рис. 2.2).

Матеріали URL: Var2

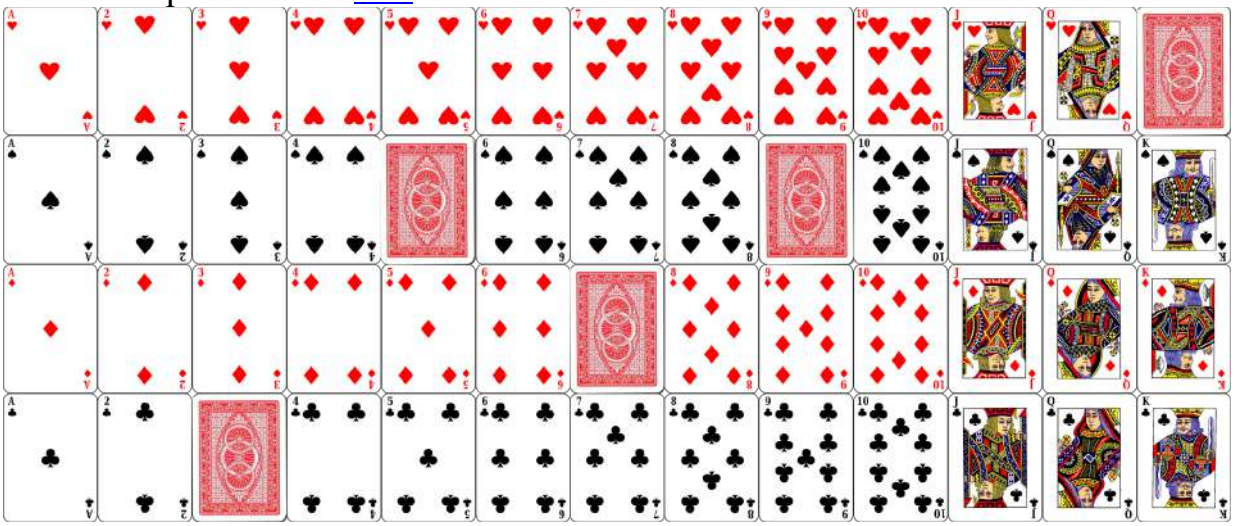


Рис. 2.2. Приклад інтерфейсу до завдання 3.

Завдання 4

Надати квадратним блокам (рис. 2.3) можливість перетягування (draggable) в межах контейнера. При наведенні курсора на блок-квадратик і утриманні кнопки миші, цей елемент повинен переміщатися. Блоки повинні залишатися повністю в межах контейнера і не повинні навіть частково виходити за його межі.

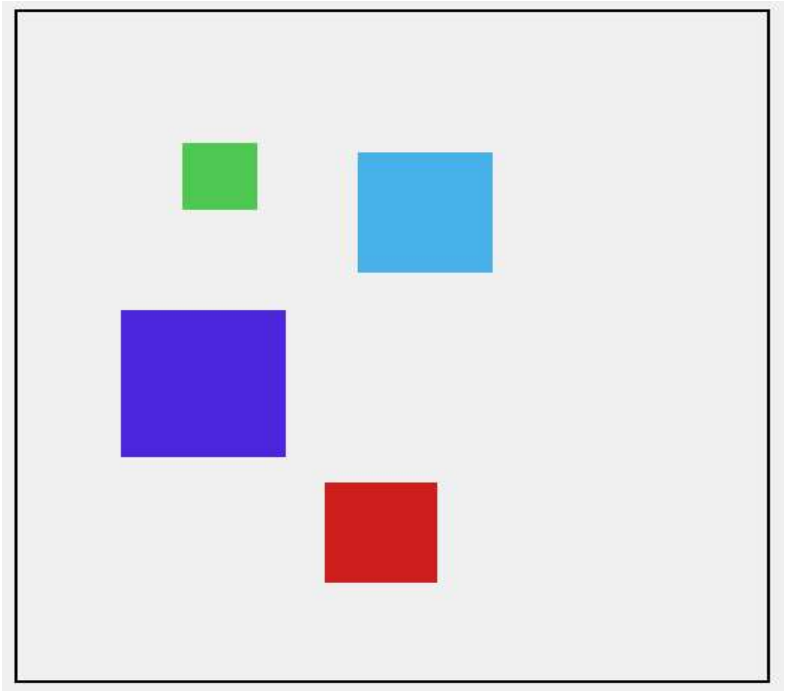


Рис. 2.3. Приклад інтерфейсу до завдання 4.

Завдання 5

Зверстайте галерею за зразком та реалізуйте підсказку, що виникає при наведенні курсору на зображення. Напис слідкує за курсором на зображенні та зникає при виході за межі.

Приклад інтерфейсу див. URL: Var4_2.png

Завдання 6

Реалізувати кнопку для вебсторінки, при натисканні на яку в контейнері будуються *n* кругів. Центр круга – випадкові координати в межах контейнера, радіус – випадкова величина від 10px до 50px. Значення n вводиться в текстовому полі. При натисканні на круг, він повинен зникати.

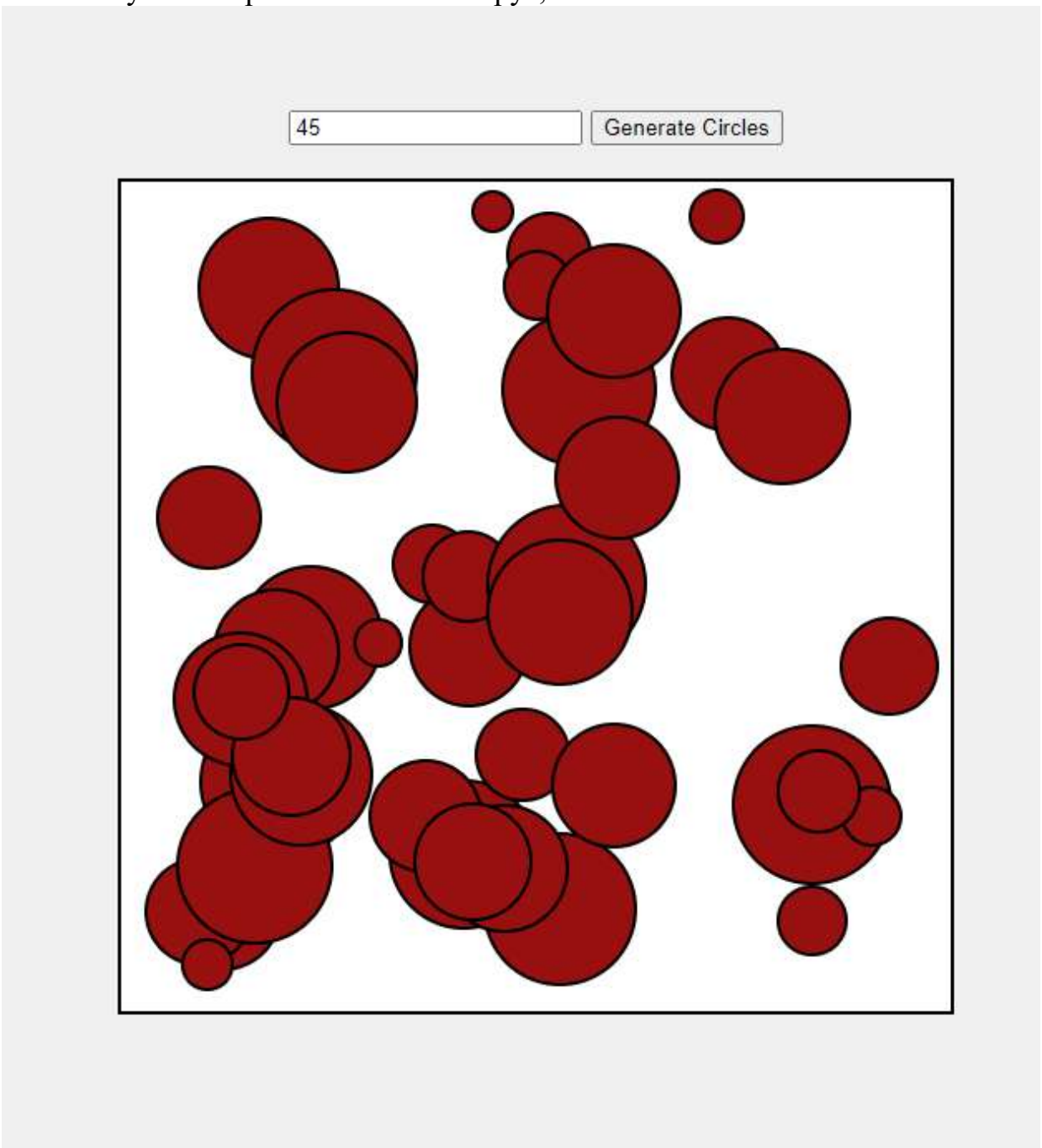


Рис. 2.4. Приклад інтерфейсу до завдання 6.

Завдання 7

Реалізувати в контейнері можливість переглядати світлини одна за одною при клацанні курсора. Світлини змінюють одна одну з ефектом слайдингу.

Рис. 2.5. Приклад інтерфейсу до завдання 7.