10.5. CSS. Стилі коду та побудова макетів

Робота з елементами HTML, інспектування та дослідження на помилки коду. Стилі написання CSS та побудова макетів. Адаптивний дизайн.

1. Особливості роботи з об'єктами веб-сторінки
    a. Текст
    b. Списки
    c. Посилання
    d. Медіафайли
    e. Таблиці
2. Виправлення CSS за допомогою інструментів розробника
3. Стилі написання коду
    a. Мінімізуйте використання препроцесорів
    b. Акуратніше з CSS-методологіями
    c. Використовуйте гнучкі, відносні одиниці
    d. Не переборщуйте зі скиданнями стилів
    e. Планування до дії
    f. Єдиний та наочний синтаксис
4. Макет CSS
    a. Нормальний потік
    b. Флоати (float, обтікання)
    c. Флексбокси (flexbox)
    d. Позиціювання
    e. Сітка (Grid)
    f. Багато колонок
    g. Адаптивний дизайн

Особливості роботи з об'єктами веб-сторінки

Крім загальних властивостей, які мають багато HTML-елементів, важливо розібратися в роботі з текстом, списками, посиланнями, формами, таблицями, медіа-файлами. Каскадні таблиці стилів надають широкий вибір інструментів у цьому напрямку.

Текст

Форматування текстового контенту здатне суттєво покращити сприйняття інформації. Текстові елементи можуть бути як рядковими (, , <i>), так і блоковими (<article>, , <aside>). Це потрібно враховувати під час роботи з ними.

Опишемо головні можливості CSS при взаємодії з текстовим вмістом:

1) Колір (задається властивістю color, за умовчанням успадковується від <body>)

Приклад - HTML

Текст параграфа набув темно-синього відтінку.

Приклад - CSS

```
p {
    color: darkblue;
```

```
}
```

2) Вирівнювання по горизонталі (використовується властивість text-align, актуальне лише для блокових елементів)

Приклад - HTML

```
<h1>Заголовок справа</h1>
<p>Текст параграфа вирівняємо по центру</p>
```

Приклад - CSS

```
p {
    text-align: center;
}
h1 {
    text-align: right;
}
```

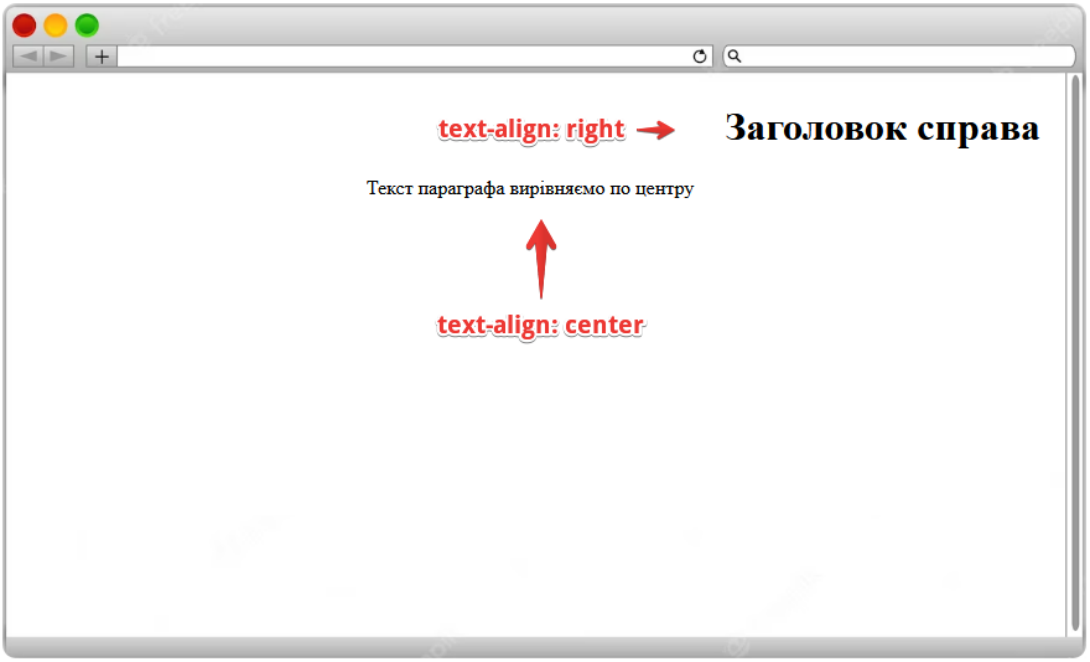


Рис.1. Вирівнювання тексту

3) Напрямок (для мов з неєвропейським напрямом написання тексту)

Приклад - HTML

```
<span>Арабський напрямок листа</span>
```

Приклад - CSS

```
p {
    direction: rtl;
    unicode-bidi: bidi-override;
}
```

4) Декорування (різні стилі підкреслення)

Приклад - HTML

```
<p>
  <span class="under">Значення</span>
```

```
  практики
  <span class="through">можна</span>
  не можна переоцінити, як кажуть
  <span class="above">лідери</span>
</p>
```

Приклад - CSS

```
.under {
    text-decoration: underline;
}
.through {
    text-decoration: line-through;
}
.above {
    text-decoration: overline;
}
```

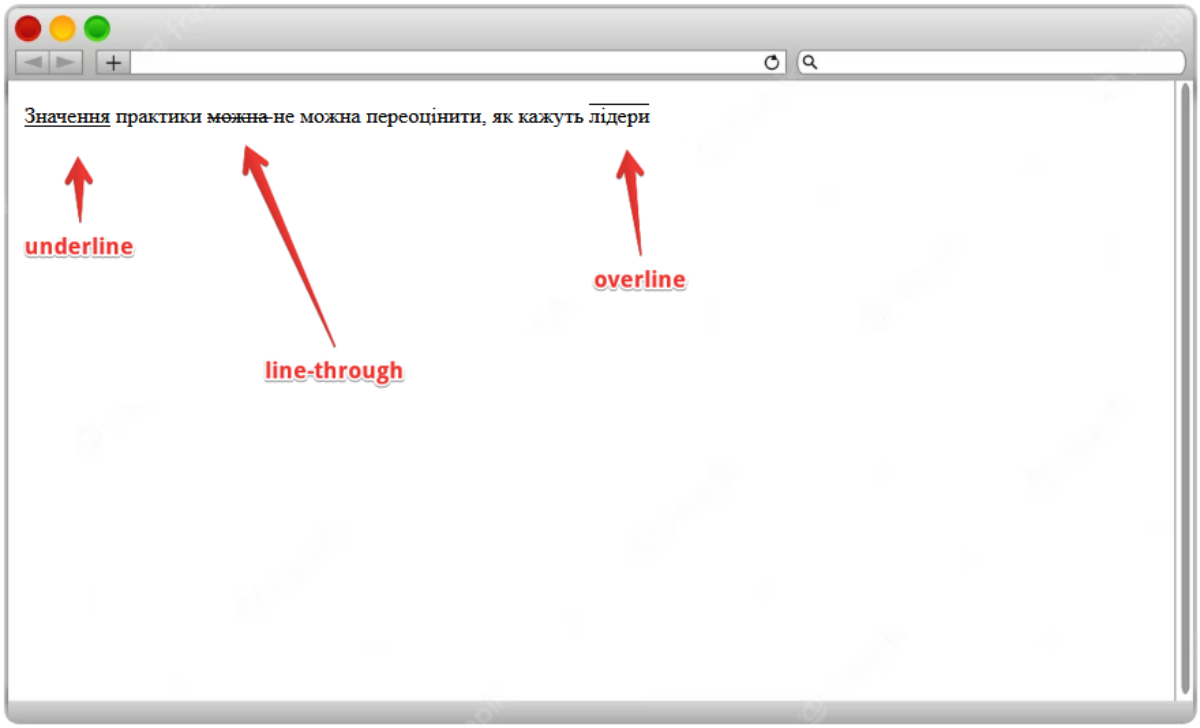


Рис.2. Різні стилі підкреслення

5) Перетворення (до верхнього або нижнього регістру, виділення першого символу)

Приклад - HTML

```
<h1>Капслоком варто позначати лише важливі невеликі фрагменти</h1>
```

Приклад - CSS

```
h1 {
    text-transform: uppercase;
}
```

6) Відступи, відстані (між рядками, літерами, словами)

Приклад - HTML

```
<p>Текстовий блок, що представлений багатьма словами, літерами та виразами. Корисна інформація. Складно структуровані речення.</p>
```

Приклад - CSS

```
p {
    width: 20vw;
    text-indent: 25px;
    letter-spacing: 3px;
    line-height: 5vh;
    word-spacing: 1vw;
    white-space: normal;
}
```

7) Вирівнювання по вертикалі

Приклад - HTML

```
<p>
     Текст одному рівні
     <span>А цей - вище</span>
</p>
```

Приклад - CSS

```
span {
    vertical-align: 33px;
}
```

8) Тіні (горизонтальні чи вертикальні зрушення, з розмиттям, конкретного кольору)

Приклад - HTML

```
<h1>Прикрашаємо заголовок</h1>
```

Приклад - CSS

```
h1 {
    text-shadow: 7px 7px 10px teal;
}
```

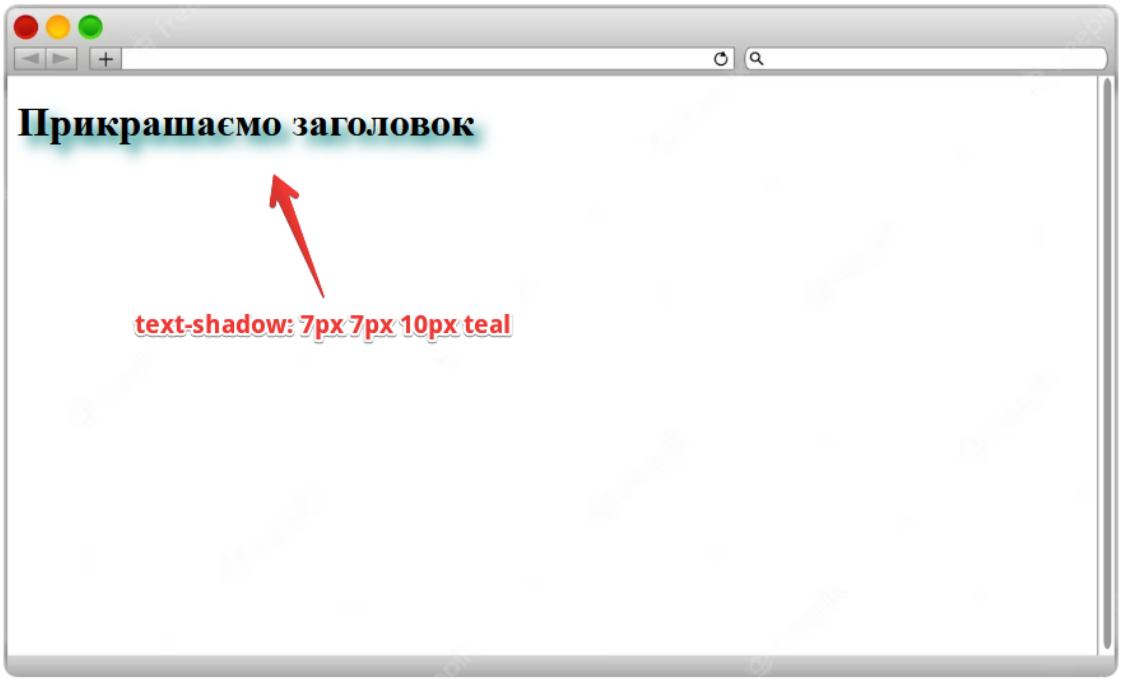


Рис.3. Заголовок з тінню

Призначення шрифтів

Для тексту можна призначати шрифти. Підключаються вони двома способами:

1. Через медіазапити.
2. Через тег <link>.

При першому варіанті в заголовній частині HTML-документа прописуються потрібні посилання, а шрифти для елементів призначаються через CSS.

Приклад - HTML

```
<head>
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link
href="https://fonts.googleapis.com/css2?family=RocknRoll+One&display=swap
" rel="stylesheet">
</head>
<body>
  <h1>Сучасний Google шрифт</h1>
</body>
```

Приклад - CSS

```
h1 {
    font-family: 'RocknRoll One', sans-serif;
}
```

У другому випадку використовуємо медіазапит.

Приклад - HTML

```
<h1>Сучасний Google шрифт</h1>
```

Приклад - CSS

```
@ import
url('https://fonts.googleapis.com/css2?family=RocknRoll+One&display=swap'
);
h1 {
    font-family: 'RocknRoll One', sans-serif;
}
```

Списки

У HTML присутні 3 типи списків: нумеровані (<ol>), ненумеровані (<ul>)та списки визначень (<dl>). Кожен із типів можна стилізувати за допомогою CSS аж до створення своїх власних маркерів з іконок або спеціальних символів.

Приклад - HTML

```
<ul>
  <li class="square">Квадрат</li>
  <li class="circle">Коло</li>
  <li class="own">Власний малюнок</li>
</ul>
<h1>Списки нумеровані</h1>
<ol>
  <li class="decimal-leading-zero">З нулем спочатку</li>
  <li class="greek">Грецька</li>
  <li class="japan">Катакана</li>
</ol>
```

Приклад - CSS

```
.circle {
    list-style-type: circle;
}
.square {
    list-style-type: square;
}
.own {
    list-style-image: url("../img/mark.png");
}
.decimal-leading-zero {
    list-style-type: decimal-leading-zero;
}
.greek {
    list-style-type: lower-greek;
}
.japan {
    list-style-type: katakana;
```

```
}
```

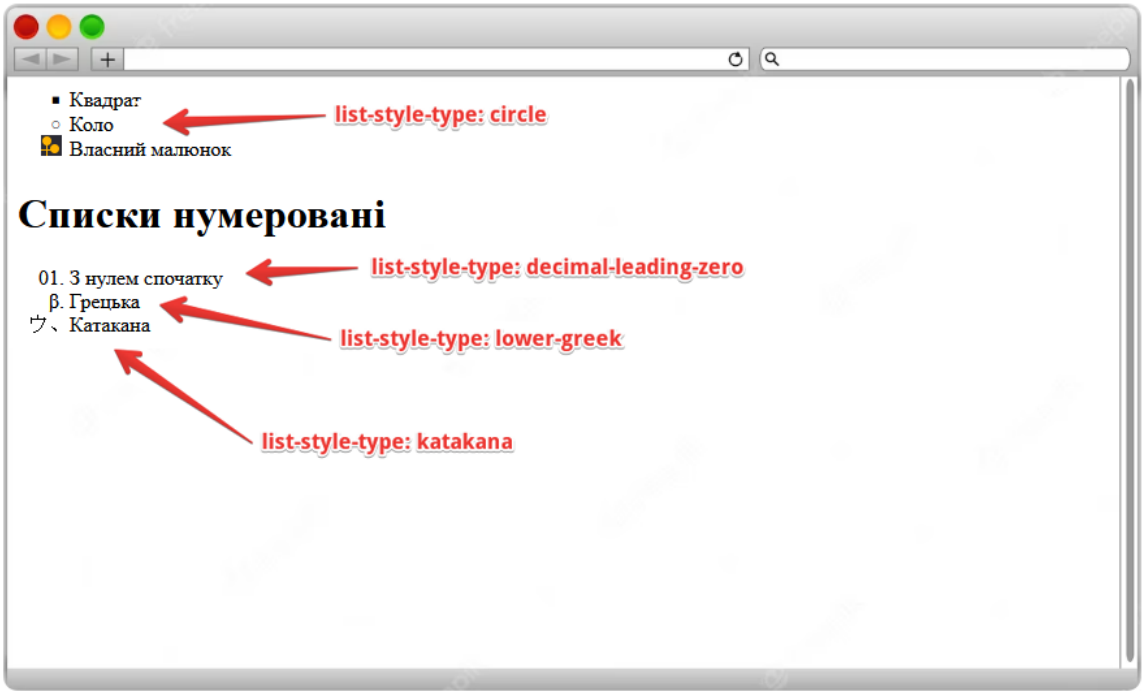


Рис.4. Різні стилі списків

Крім списків можуть застосовуватися лічильники (counters).

Приклад - HTML

```
<h1>Лічильники</h1>
<ol>
  <li>Елемент</li>
  <li>Елемент</li>
  <li>Елемент
      <ol>
          <li>Внутрішній елемент</li>
          <li>Внутрішній елемент</li>
      </ol>
  </li>
  <li>Елемент</li>
</ol>
```

Приклад - CSS

```
ol {
    counter-reset: section;
    list-style-type: none;
}

li ::before {
    counter-increment: section;
    content: counters(section, ".")"--";
}
```

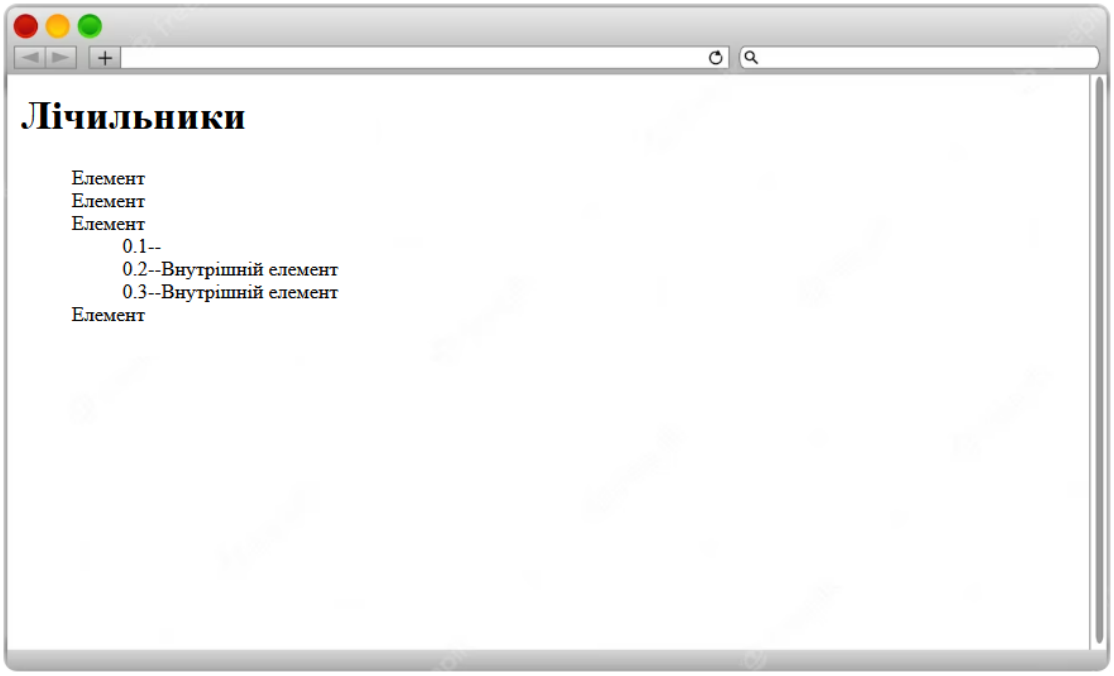


Рис.5. Приклад лічильника

Посилання

За замовченням стиль посилань практично у всіх браузерах однаковий: вони підкреслені та виділені синім кольором. Після переходу за посиланням його колір змінюється на фіолетовий. Таку поведінку, звісно, можна змінювати. Часто посилання надають у вигляді кнопок.

Приклад - HTML

```
<h1>Посилання</h1>
<a href="#" class="custom">Прикрашаємо посилання</a>
<a href="#" class="icon">Посилання з іконкою</a>
<a href="#" class="button">Посилання у вигляді кнопки</a>
```

Приклад - CSS

```
a {
    outline: none;
    text-decoration: none;
    color: rgb(62, 129, 131);
    display: block;
    margin: 1rem;
}
.custom:hover {
    text-decoration: line-through;
    color: darkcyan;
    font-size: 1.2rem;
}
.icon:hover {
    color: #000;
    text-transform: uppercase;
```

```
    text-decoration: underline blueviolet;
}
.icon:hover::before {
    content: url("../img/mark.png");
}
.button {
    width: 200px;
    height: 40px;
    background-color: gold;
    text-align: центр;
    line-height: 2;
    border-radius: 20%;
}
.button:hover {
    background-color: goldenrod;
}
```

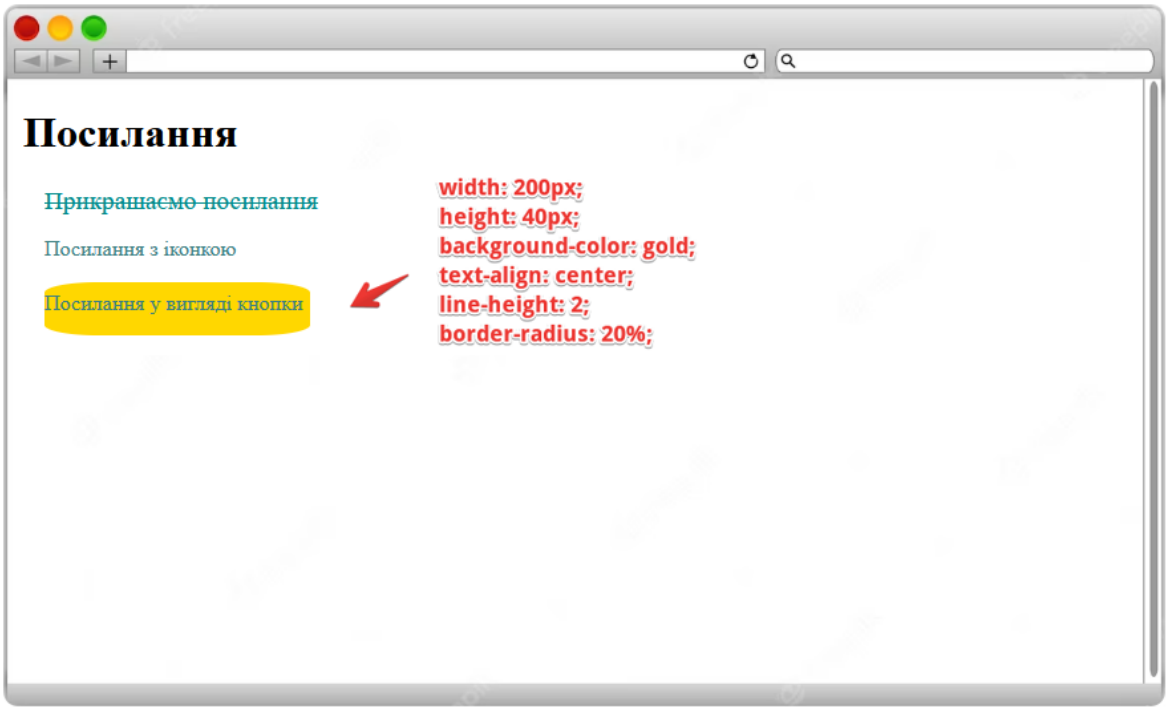


Рис.6. Різні стилі посилань

Медіафайли

За допомогою CSS можлива стилізація картинок, відео або аудіо. Для них визначаються розміри, внутрішні та зовнішні відступи, межі, способи заповнення доступного простору.

Приклад - HTML

```
<h1>Зображення</h1>
<div>
  <img src="../img/4.png" alt="Icon">
</div>
```

```
<div>
  <img src="../img/1.png" alt="Iryna">
</div>
```

Приклад - CSS

```
div {
      width: 300px;
      height: 300px;
      border: darkorange 3px solid;
      display: inline-block;
}
img {
    height: 100%;
    width: 100%;
}
img [alt="Icon"] {
    object-fit: cover;
}
img [alt="Iryna"] {
    object-fit: contain;
}
```

За допомогою властивості object-fit можна змусити зображення заповнити доступний простір максимально ефективно без спотворення його розмірів.

Рис.7. Заповнення картинками блоків div

Таблиці

Конкретні властивості реально поставити як для всієї таблиці, так окремих її рядків і стовпців. Тут можна оперувати кольорами, розмірами, шрифтами, відступами.

Приклад - HTML

```
<h1>Таблиці</h1>
<table>
  <tr>
      <td>1</td>
      <td>2</td>
      <td>3</td>
      <td>4</td>
  </tr>
  <tr>
      <td>10</td>
      <td>20</td>
      <td>30</td>
      <td>40</td>
  </tr>
</table>
```

Приклад - CSS

```
table {
    width: 80vw;
    background-color: darkseagreen;
    border: darkviolet 1px solid;
}
table tr:first-child {
    background-color: deepskyblue;
}
table tr td:last-child {
    background-color: thistle;
    font-size: 2rem;
    text-align: center;
}
```

Для першого рядка таблиці визначено особливий колір фону, а останніх комірок у кожному рядку змінено розмір шрифту, колір фону і вирівнювання тексту.

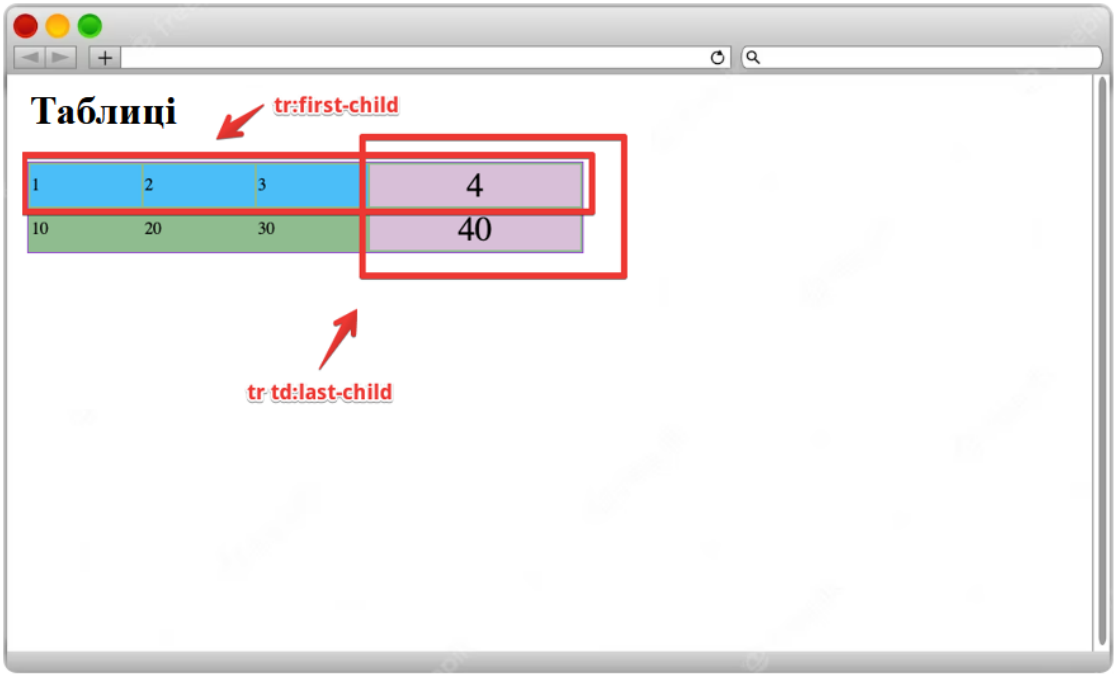


Рис.8. Використання таблиці

Виправляння CSS за допомогою інструментів розробника

У будь-якому сучасному браузері є інструменти розробника (часто викликаються клавішею F12). З їхньою допомогою перевіряють окремі ділянки розмітки, вивчають властивості елементів, виявляють помилки в коді CSS (коли щось працює не так, як заплановано).

Приклад - HTML

```
<article>
      <header>Виправляємо CSS</header>
      <main>
            Використовуємо браузер Chrome.
            <span>Задамо висоту блоку 300 пікселів</span>
      </main>
      <footer>Посилання на матеріал є обов'язковим</footer>
</article>
```

Приклад - CSS

```
span {
      height: 300px;
      background: red;
      color: white;
}
```

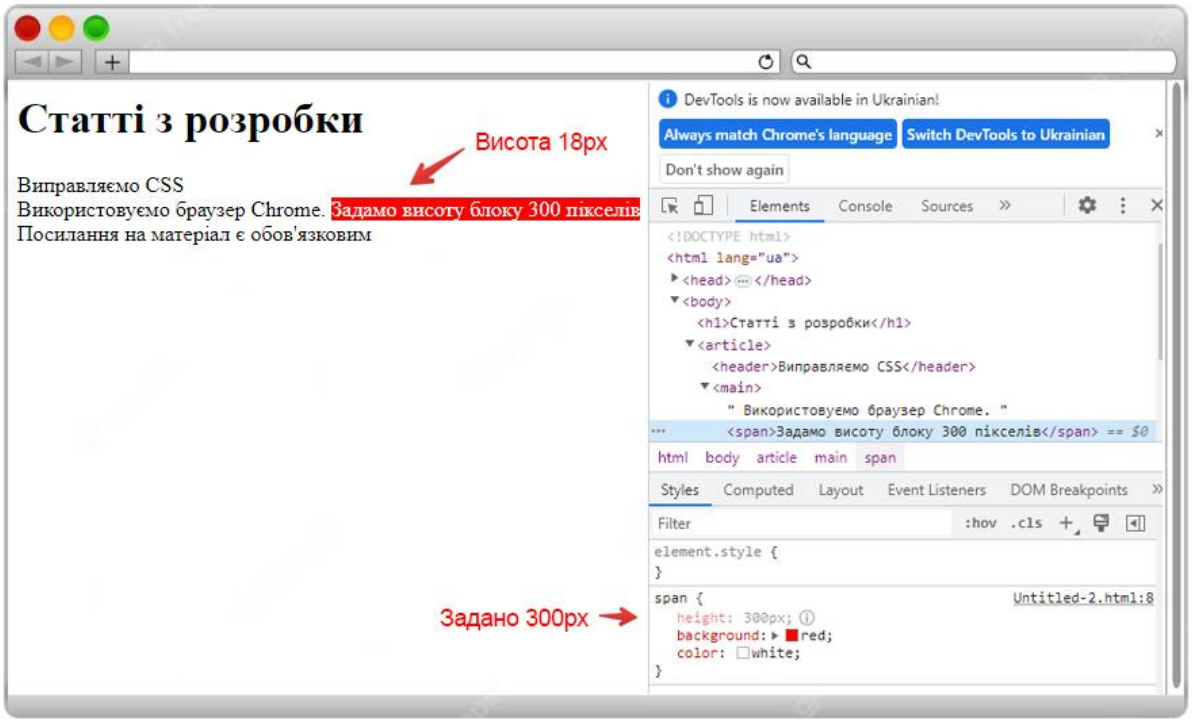


Рис.9. Блок span із заданою висотою 300px

Для розуміння цієї проблеми зайдемо в розділ Computed. Тут бачимо як привласнені розробником атрибути, та ті, які браузер сам визначив виходячи з налаштувань чи батьків блоку.

Можна побачити властивість display: inline. Навіть якщо ми забули, що якийсь елемент сайту відображається як рядок (для якого задавати висоту або ширину марно), на панелі розробника це легко з'ясувати.

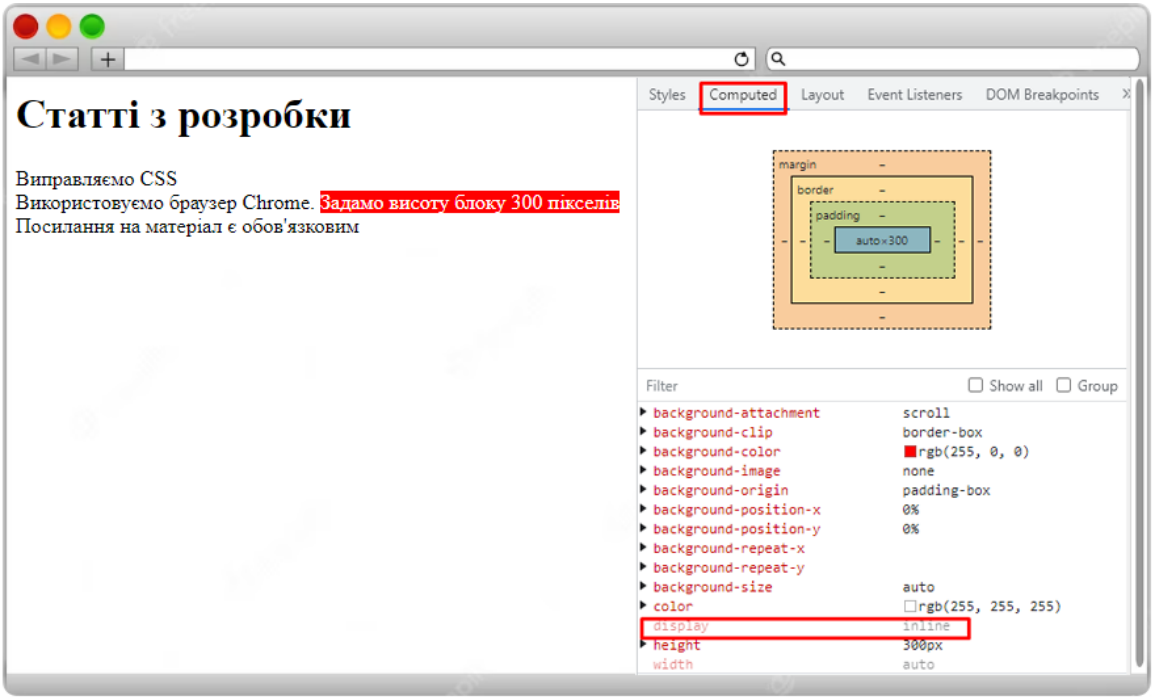


Рис.10. До блоку span не застосувалася висота, тому він - рядковий

Зробимо наш елемент span блоковим - застосуємо властивість display: block.

Приклад - CSS

```
span {
    height: 300px;
     background: red;
     color: white;
     display: block;
}
```

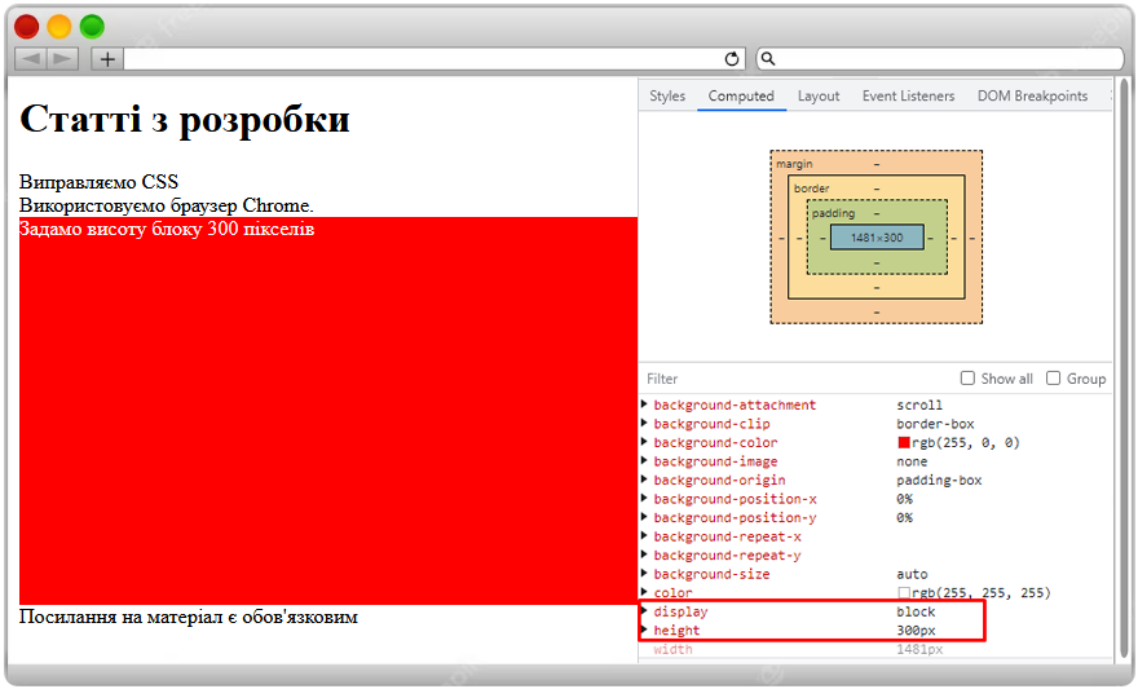


Рис.11. Елемент span тепер блоковий - висота 300px

Хоч приклад і примітивний, але показує, наскільки простіше можна досліджувати і правити стилі в браузері, ніж за допомогою копирсання в нескінченному коді файлу CSS. У реальній верстці може бути кілька файлів CSS, багато документів HTML, в яких легко заплутатися. Інструменти розробника дозволяють полегшити модифікацію стилів, а також подивитися з якого конкретного файлу вони привласнені.

В панелі розробника не просто досліджують властивості елементів або виявляють помилки, але й можна протестувати інші атрибути, прямо прописавши їх, щоб зрозуміти, наскільки вони відповідають задуму дизайнера.

Стилі написання коду

Розробляючи навіть невеликий проект поодинці, не варто забувати про Style Guide. Хоча може здатися, що вам все зрозуміло і ви легко читаєте написане, через якийсь час неструктуровані CSS-документи почнуть викликати головний біль. А у випадках зайнятості масштабними проектами індивідуальність зовсім не вітається. Чи хотілося б особисто вам розбиратися в чужому коді, написаному суцільним неструктурованим набором без якоїсь єдиної стилізації?

Для вирішення цієї проблеми є домовленості, правила, стилі написання коду. Вони є й у CSS. Комп'ютеру або браузеру зовсім не важливо, як написані документи верстальника, але для людини це має значення.

Рекомендується дотримуватись таких правил:

Мінімізуйте використання препроцесорів

Для спрощення розробки CSS придумано препроцесори (Sass, Less, SCSS). Вони перетворюють опис властивостей стилів в окрему мову програмування, істотно прискорюючи процес верстки. Якщо всі учасники команди знайомі з цим інструментом, то ним можна і потрібно користуватися.

У будь-якому випадку це підвищує рівень вимог до кандидатів на посаду верстальника в компанію. Максимум простоти та ясності, а також мінімальний поріг входу – гарна практика.

Приклад – Sass

```
$fonts: Roboto, sans-serif
$main-color: #333
body
  font: 100% $fonts
  color: $main-color
```

Приклад - CSS

```
body {
    font: 100% Roboto, sans-serif;
    color: #2ff3aa;
}
```

Вище наведено код на Sass та його аналог у CSS. Другий приклад зрозуміє кожен, хто хоч якось стикався з мовою CSS, а ось розібратися в коді Sass не кожен зможе без відповідної підготовки.

Препроцесори не заборонені, але актуальні лише там, де є досвідчені розробники, які розуміють їх синтаксис.

Акуратніше з CSS-методологіями

Методики, про які сказано нижче, не модифікують CSS-мову. У цьому їх застосування вимагає деякої підготовки. У великих проектах вони використовуються повсюдно, але для невеликих проектів не завжди необхідні. Вони зручні, зрозумілі знаючим верстальникам, але не початківцям.

Спочатку необхідно домовитися і досягти єдиного рівня розуміння методології, а вже потім впроваджувати її в розробку. Найбільш відомими є такі підходи:

1. BEM.
2. OOCSS.
3. Atomic.
4. SMACSS.

1) BEM (Block Element Modifier)

Є правилами на призначення імен. Це важлива річ у будь-якій мові програмування. Ім'я класу чи ідентифікатора можна встановити випадковим набором символів (тоді ніхто не побачить його логіку) або осмислено (що спростить розуміння).

Наведемо приклад:

- navbuttonactive
- NavButtonActive
- nav-button-active

У першому випадку зрозуміти сенс імені не просто: потрібно придивитись, щоб виділити окремі слова. Другий і третій варіанти наочніші, але, не дають ясності щодо того, з чим ми маємо справу: блоком, елементом або модифікатором.

Саме для цього в методології BEM запроваджено ці 3 категорії:

Блок

Незалежна автономна сутність. Це деякі базові одиниці сайту (меню, заголовки, шапка тощо). Вони складаються з одного або кількох слів (через дефіс), що задають ім'я класу.

Типові приклади іменування:

- class="navigation"
- class= "main-page-article"

Елемент

Семантично прив'язані до окремих блоків і не мають незалежного стану як зрозуміла одиниця HTML-документа. Привласнення імені здійснюється через клас із зазначенням імені блоку, а через 2 нижні підкреслення – назви елемента.

Приклад

```
class="article__header"
```

Тут одразу зрозуміло, що ми звертаємось до заголовків статей.

Модифікатор

Конкретні стани блоків чи модифікаторів залежно від ситуації. Імена задаються починаючи з двох дефісів після назви блоку чи елемента.

Приклад

```
class="nav__button--active"
```

Тут звертаємось до активної кнопки навігаційного меню сайту.

2) SMACSS (масштабована і модульна архітектура для правил каскадних таблиць стилів)

У цій методології йдеться про структурування CSS-документів. Як роблять зазвичай недосвідчені верстальники: у міру модифікування елементів сторінок у CSS-файл дописуються певні стилі. Розібратися згодом, у міру збільшення розміру сайту, у всьому цьому нагромадженні коду досить складно, особливо коли потрібно внести корективи.

Засновники технології пропонують наступну структуру CSS-документу:

- Базові правила (пов'язані із сайтом загалом). Тут прописують стилі для основних блоків сторінки: тіла, кнопок, списків тощо.

- Правила макета (здійснюється робота з блоками, які зустрічаються на сторінці в єдиному екземплярі. Зручніше позначати їх через ідентифікатори). Сюди входить шапка сайту, бічна панель, футер.
- Стилі модулів (масивні блоки у кількох примірниках). Вони містять: окремі статті, оформлення зображень тощо.
- Правила стану (залежно стану об'єкта його поведінка може змінюватися: при наведенні мишею, виділенні, фокусуванні тощо).
- Оформлення (прикраси, здатні змінюватися з часом). Наприклад, новорічна тема, ювілей сайту та інше.

Таким чином, застосування методологій значно спрощує роботу з кодом, але потребує певної підготовки.

Використовуйте гнучкі, відносні одиниці

Розмірів моніторів сьогодні кілька десятків, не кажучи про типи мобільних пристроїв. Щоб під кожен випадок підібрати оформлення сайту (особливо розміри блоків) може знадобитися багато часу. Набагато простіше застосовувати відносні чи відсоткові значення у міру можливостей.

Приклад - CSS

```
p {
    font-size: 12 px;
}
p {
    font-size: 1.3rem;
}
```

Розмір шрифту в першому параграфі заданий абсолютним значенням (і на великих розмірах екрану прочитати його буде не просто), а ось другий варіант гнучкіший (шрифт буде підлаштовуватися під базові налаштування користувача).

Не зловживати зі скиданнями стилів

Коли ми поміщаємо текст у тег <p>, наприклад, цей блок за замовчуванням має зовнішні відступи. Але ми їх не ставили. Браузер сам так вирішив відповідно до своїх налаштувань. Це може призвести до ряду проблем при макеті.

Саме тому розробники люблять скидати стилі всіх елементів до нульових значень, а потім задавати ті атрибути, які їм цікаві. Є й мінус такого підходу: багато зайвої писанини.

Ніколас Галлахер спеціально для такого випадку написав файл скидання CSS-стилів normalize.css, який потрібно підключати до веб-сайту перед власними стилями. Не варто забувати, що не завжди може знадобитися скасувати налаштування браузера. Але якщо потрібно відстежувати кожен атрибут самостійно, то краще використовувати normalize.css.

Планування до дії

Перш ніж почати заповнювати CSS-документ своїми стилями, ретельно продумайте структуру макета та стилістику окремих елементів. Це дозволить скоротити код та докласти менше зусиль.

Якщо пропустити крок планування, робота верстальника відштовхуватиметься від взаємодії з конкретним елементом, а в міру просування сторінкою він багаторазово повторить себе, хоча міг би оптимізувати розмітку в рази.

Єдиний та наочний синтаксис

Стилістика коду призводить до ясності при повторному перегляді документів та внесенні правок. Значно простіше працювати з красиво оформленим CSS-файлом, що має структуру та наповнений коментарями. Це стандарт за замовченням, без якого вас ніхто не вважатиме грамотним верстальником.

Приклад - CSS

```
/* Варіант 1 */
section {
    font-size: 2em;
    color: red;
    text-align: justify;
}
/* Варіант 2 */
section {font-size: 2em; color: red; text-align: justify;}
```

Браузер не скаже відвідувачу сайту ні слова, тому що зрозуміє і перший варіант стилізації розділу, і другий. Але програмісту, що працює з таким файлом, останній спосіб написання правил зовсім не сподобається.

Потрібно дотримуватися наочності та читабельності коду. Не варто забувати і про коментарі, щоб у міру повернення до роботи над проектом не витрачати час у пошуках потрібного коду. Також рекомендується уникати скорочених описів правил за можливості, оскільки вони менш зрозумілі. У CSS значення атрибутів прийнято писати переважно в подвійних лапках, хоча браузер чудово розуміє і одинарні.

Приклад - CSS

```
/* Налаштування фону головного блоку */
/* Варіант 1 */
div {
    background-color: tomato;
    background-image: linear-gradient(#d66, #549);
    background-repeat: no-repeat;
    background-clip: padding-box;
    background-origin: padding-box;
    background-position-x: center;
    background-position-y: top;
    background-size: 70%;
    background-attachment: fixed;
}
/* Варіант 2 */
```

```
div {
    background: tomato linear-gradient(#d66, #549)no-repeat padding-box padding-box center top / 70% fixed;
}
```

У першому способі завдання фону використано значно більше рядків коду, але ні в кого не виникне проблем з розумінням кожного рядка. Скорочений варіант менш наочний і не відразу проінтерпретується навіть досвідченим верстальником.

Макет CSS

Робота з окремими HTML-елементами зазвичай не викликає жодних проблем щодо CSS. Коли йдеться про верстку макета сторінки, виникають проблеми. Блоки починають роз'їжджатися, накладатися один на одного, зміщуватись у незрозумілому напрямку, відображатися не так, як ми задумали.

Все залежить від CSS-макета та інструментарію верстки. Виділяють наступні можливості: нормальний потік, флексбокси, сітки, флоати, позиціонування, багато колонок. Важливо враховувати питання адаптивності макетів під різні пристрої.

Нормальний потік

У міру появи елемента на сторінці (відповідно до коду HTML-файлу) відображаються і його CSS-властивості. Напрямок цього процесу наступний: зверху-вниз, зліва-направо.

Кожен блоковий елемент (незалежно від розміру) займає всю ширину рядка. Навіть якщо розташувати два теги один за одним з шириною в 100px кожен (при загальній ширині батька в 1200px), вони не розташуються на одному рядку, а будуть знаходитися один під одним.

Рядкові елементи не переносяться на новий рядок, поки мають вільний простір праворуч, а йдуть один за одним.

Якщо потрібна інша поведінка елементів, його можна змінити. Наведемо приклади.

Приклад - HTML

```
<h1>Нормальний потік</h1>
<div>
     <article><strong>Стаття</strong>номер <i>один</i></article>
     <article>Ще <em>одна</em> стаття</article>
</div>
```

Приклад - CSS

```
article {
    width: 200px;
    height: 130px;
    border: saddlebrown 3px solid;
}
```

Як видно, окремі статті йдуть одна під одною, тому що кожен блоковий елемент займає всю ширину рядка.

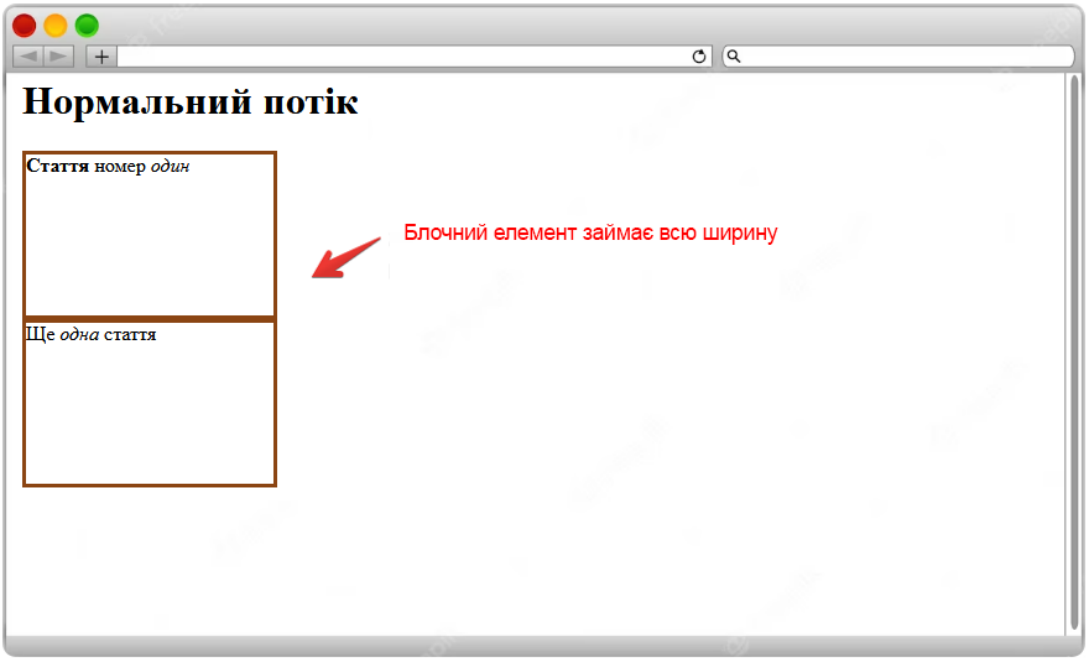


Рис.12. Блоковий елемент займає всю ширину рядка

Флоати (float, обтікання)

Одна з перших спроб реалізувати багато колонок у макетах (не рахуючи табличного стилю, який сьогодні вважається застарілим). Спочатку флоати призначалися для картинок всередині тексту, щоб він їх обтікав з різних боків, не залишаючи порожнього простору. Можуть порушувати нормальний потік.

Приклад - HTML

```
<h1>Флоати</h1>
<section>
   <article>Я ліворуч</article>
   <article>Я праворуч</article>
   <article>Я по центру</article>
   <article>Я сам по собі</article>
</section>
```

Приклад - CSS

```
section {
    border: saddlebrown 3px solid;
}
article {
    height: 100px;
    border: brown 3px dotted;
    margin: 10px;
    padding: 15px;
}
section article:nth-child(1) {
    float: left;
```

```
}
section article:nth-child(2) {
    float: right;
}
section article:nth-child(4) {
    clear: both;
}
```

Якщо елементу задати ліве обтікання, він сам розташуватися в лівому краю, тоді як наступні елементи обтікатимуть його праворуч (і навпаки).

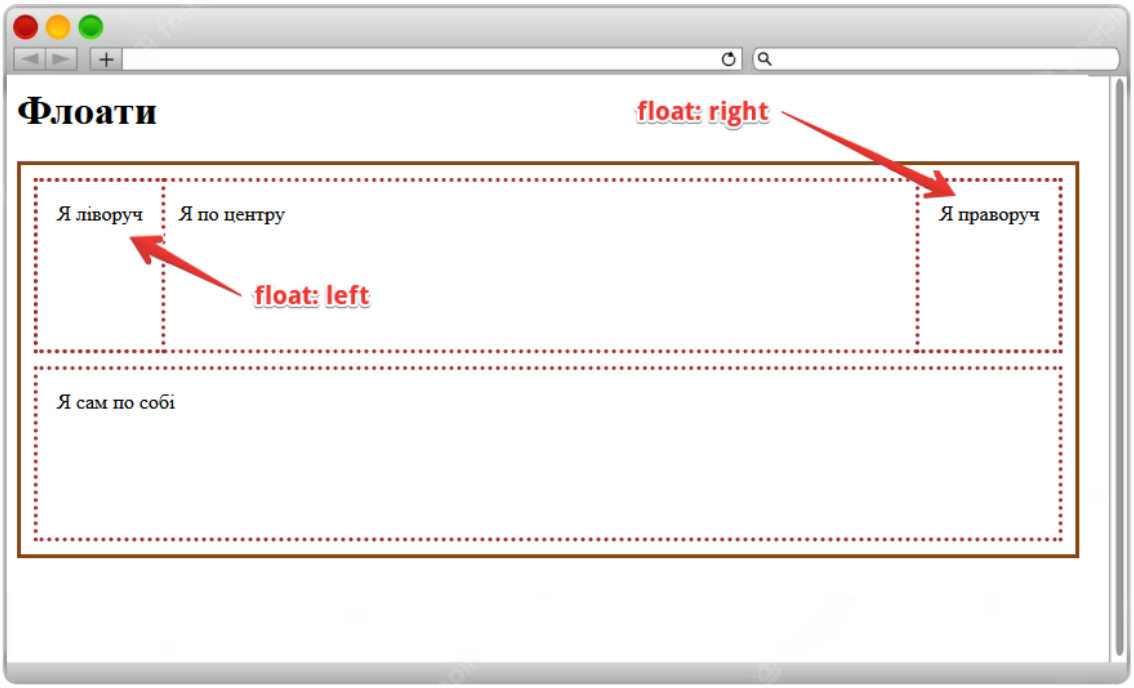


Рис.13. Розміщення блоків із застосуванням `float`

Щоб прибрати ефект для сусідніх сусідів, у першого з них потрібно визначити властивість clear: both. Це загальноприйнятий спосіб роботи із флоатами. Якщо елементи не розміщуватимуться по ширині вікна браузера, вони почнуть переноситися на новий рядок.

Флексбокси (flexbox)

Метод макетування, що розміщує елементи в колонках або рядках. Об'єкти розтягуються або стискаються, щоб зайняти доступний для них простір. Технологія, що дозволяє досягти того, чого раніше не дозволяли зробити флоати і позиціонування:

- Вертикально відцентрувати елемент всередині батька.
- Поступово зайняти доступний простір батька кількома нащадками.
- Задати однакову висоту блокам, навіть якщо їх контент відрізняється за обсягом.

При взаємодії з флексбоксами необхідно засвоїти термінологію:

- Є головна і перпендикулярна їй осі (напрямок головної осі визначається властивістю flex-direction).
- flex-контейнер – батько flex-елементів, на які поширюються властивості гнучкості.

Опишемо властивості флексбоксів:

Властивість	Можливі значення	Опис
flex-direction	column, row, row-reverse, column-reverse	Задає напрямок головної осі
justify-content	flex-start, flex-end, центр, space-between, space-around	Вирівнює елементи по основній осі
align-items	flex-start, flex-end, center, stretch, baseline	Вирівнювання об'єктів по другорядній осі
align-self	start, flex-end, safe center та ін.	Відповідає за вирівнювання окремого елемента контейнера
flex-basis	auto, будь-які одиниці довжини	Розмір елемента по осі
flex-grow	числовий коефіцієнт	Перевага у займаній площі конкретного елемента порівняно з рештою
flex-shrink	числовий коефіцієнт	Стиснення блоку зі зменшенням простору батька
flex-wrap	wrap, nowrap, wrap-reverse та ін.	Чи дозволяти перенос блоків або використовувати один рядок з внесенням горизонтального прокручування

Приклад - HTML

```
<h1>Флексбокси</h1>
<section>
   <article>Вчимося програмувати</article>
  <article>Сила флексбоксів</article>
   <article>Як розтягнути елементи на всю ширину екрана</article>
   <article>У розробці...</article>
</section>
```

Приклад - CSS

```
section {
```

```
    border: saddlebrown 3px solid;
    width: 90vw;
    height: 70vh;
    background-color: burlywood;
    display: flex;
    align-items: center;
    justify-content: space-around;
    flex-direction: row-reverse;
}
article {
    height: 30%;
    width: 15%;
    border: brown 3px dotted;
    margin: 10px;
    padding: 15px;
}
```

Всі елементи розташовані горизонтально у зворотному порядку, між ними простір розподілено порівну, а також присвоєно вирівнювання по центру другорядної осі.

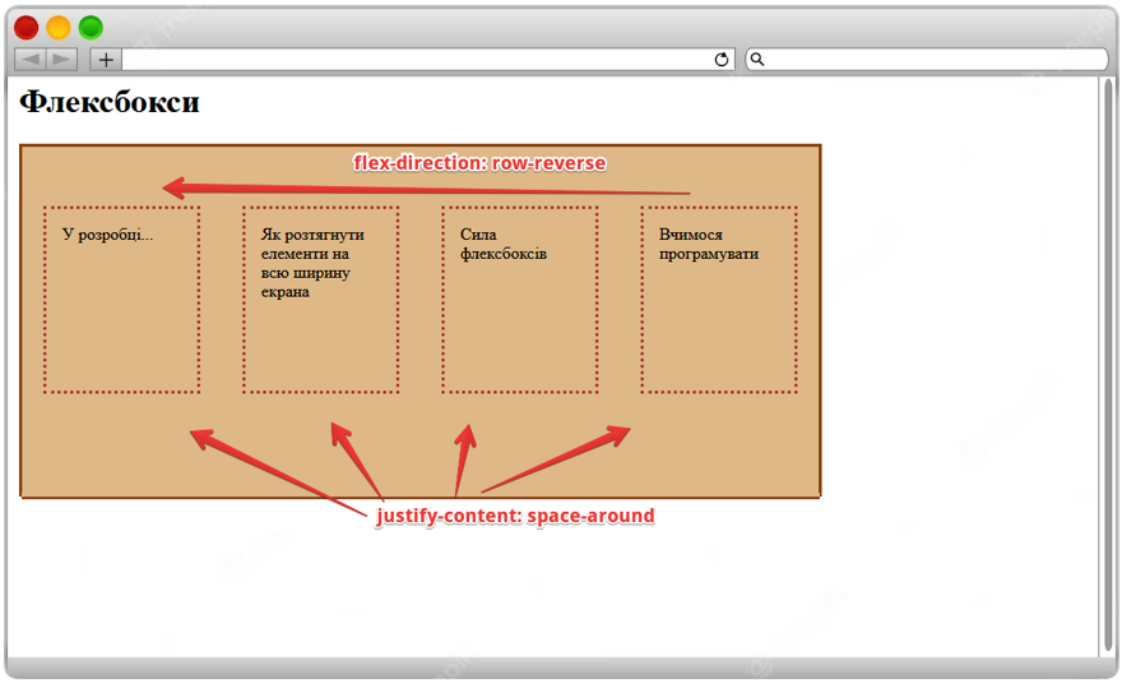


Рис.14. Розташування блоків у режимі Flexbox

Позиціювання

За допомогою позиціонування можна витягувати елементи з нормального потоку і розташовувати так, як вимагає завдання. Вони можуть бути один на одному, а то й зовсім не рухатися при прокручуванні сторінки.

Властивість position відповідає за цю методику. Можливі значення:

Значення	Опис
absolute	Інші об'єкти на сторінці поводяться так, ніби цього елемента немає. Веде відлік або від краю вікна браузера або від меж батьківського елемента
relative	Переміщає об'єкт щодо його первісного положення у нормальному потоці
static	Об'єкт повертається до нормального потоку без будь-яких додаткових ефектів.
fixed	При прокручуванні сторінки об'єкт нікуди не зміщується, а залишається на своєму місці відносно вікна браузера
sticky	Прилипає до країв робочого вікна в міру прокручування та появи в області видимості.

Приклад - HTML

```
<h1>Позиціонування</h1>
<section>
   <article>Абсолютний</article>
   <article>Фіксація</article>
   <article>Прокручування</article>
</section>
```

Приклад - CSS

```
section {
    border: saddlebrown 3px solid;
    width: 60vw;
    height: 170vh;
    background-color: burlywood;
    position: relative;
}
article {
    height: 200px;
    width: 200px;
    border: brown 3px dotted;
    margin: 10px;
    padding: 15px;
}
```

```
section article:nth-child(1) {
    position: absolute;
    top: 55px;
    right: 40px;
    background-color: forestgreen;
}
section article:nth-child(2) {
    position: fixed;
    top: 55px;
    right: 20px;
    background-color: lavenderblush;
}
section article:nth-child(3) {
    position: sticky;
    top: 45px;
    background-color: rgb (191, 176, 231);
}
```

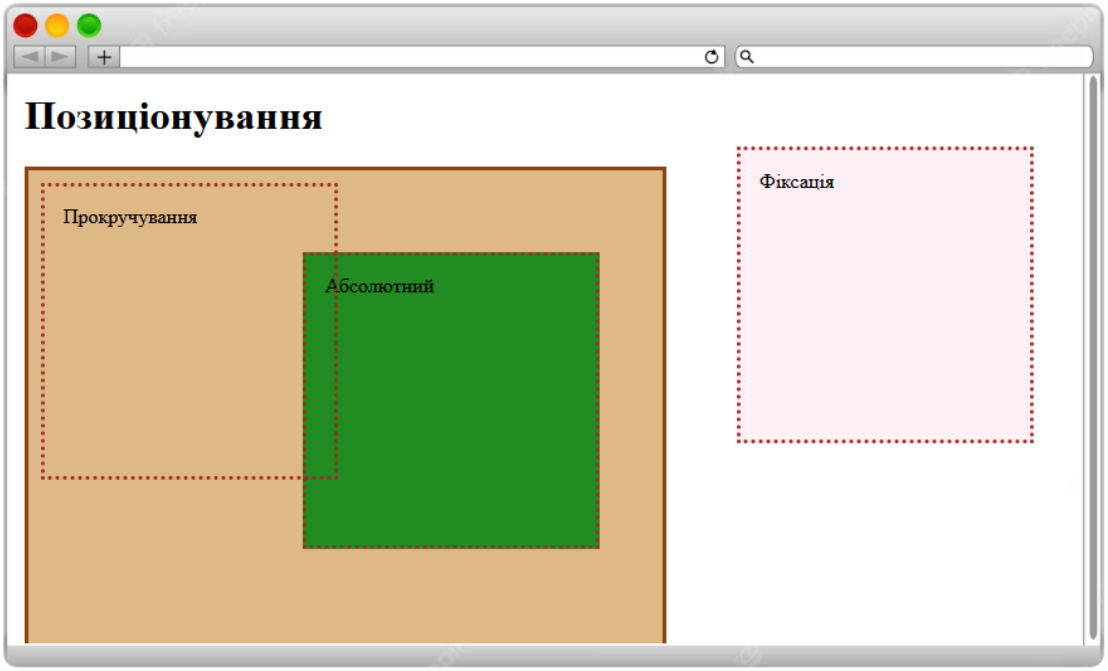


Рис.15. Блоки з різним позиціонуванням: absolute, fixed та sticky

Сітка (Grid)

Сітки сприяють поданню макета сторінки у вигляді набору колонок та рядів.

Сіткова структура передбачає наявність 3 категорій понять:

- Колонка (column).
- Ряд (row).
- Жолоб (gutter).

Щоб перетворити нащадків елемента на представників сіток, потрібно задати таку властивість: display: grid.

Основні властивості та функції, що застосовуються до гридів:

Властивість	Опис
`grid-template-columns`	Вказує ширину кожної колонки, у тому числі за допомогою нової одиниці вимірювання fr. Допустимо застосування функції repeat()
`grid-template-rows`	задає висоту ряду
`grid-auto-rows, grid-auto-columns`	Очевидне завдання розміру треків. Використання функції minmax() дозволяє масштабувати параметри комірок.
`grid-column-start, grid-column-end, grid-row-start та grid-row-end`	Для зміни обсягів займаного простору окремими комірками (об'єднання рядків чи стовпців)
`grid-column-gap та grid-row-gap`	Створення зазорів між межами сітки

Приклад - HTML

```
<h1>Система сіток</h1>
<div>
   <section>Один великий</section>
   <section>Малий</section>
   <section>Малий</section>
   <section>Малий</section>
</div>
```

Приклад - CSS

```
div {
    border: saddlebrown 3px solid;
    width: 90vw;
    height: 70vh;
    background-color: burlywood;
    display: grid;
    grid-template-columns: repeat(3, 1 fr);
    grid-auto-rows: 130px;
    grid-column-gap: 1%;
```

```
    grid-row-gap: 1rem;
}
section {
    background-color: lightslategrey;
    border: rgb(48, 33, 48)1px solid;
    padding: 10px;
}
div section:first-child {
    grid-column-start: 1;
    grid-column-end: 4;
    grid-row-start: 1;
    grid-row-end: 2;
}
```

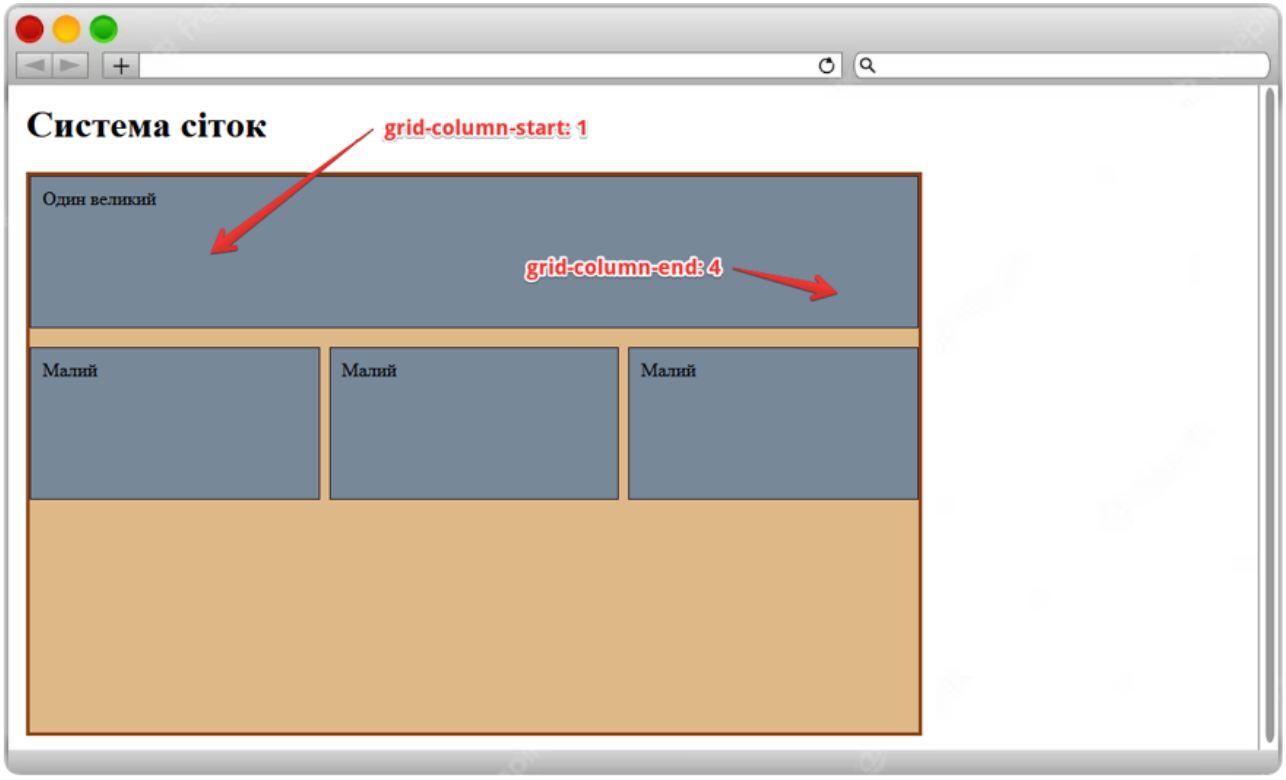


Рис.16. Розташування блоків у режимі Grid

Багато колонок

Ця стилістика макетів характерна для друкованих видань. Якщо сайт наповнений величезною кількістю текстової інформації, для зручності читання відвідувачами можна подумати про реалізацію верстки з багатьма колонками.

Основні властивості:

- column-count - задає кількість колонок;
- column-width – ширина колонки за умовчанням;
- column-gap - Відстань між стовпцями;
- column-rule – лінійка між стовпцями;
- column-span – розтягує елемент на всю ширину стовпців, розриваючи їх.

Приклад - HTML

```
<h1>Мультиколонки</h1>
<section>
   <article>На всю ширину</article>
   <article> Ця стилістика макетів характерна для друкованих видань. Якщо сайт наповнений величезною кількістю текстової інформації, для зручності читання відвідувачами можна подумати про реалізацію мультиколонної верстки. Бажання досягти зручного користування сайтом будь-якими відвідувачами з різних пристроїв призвело до поняття адаптивності верстки. Вона передбачає зміну зовнішнього вигляду ресурсу, його структури, оформлення залежно від умов.</article>
</section>
```

Приклад - CSS

```
section {
    border: saddlebrown 1px solid;
    width: 90vw;
    background-color: burlywood;
    column-count: 5;
    column-width: 100px;
    column-gap: 1rem;
}
article {
    background-color: lightslategrey;
    border: rgb(48, 33, 48)1px solid;
    padding: 10px;
}
section article:first-child {
    column-span: all;
    text-align: center;
    background-color: palevioletred;
}
```

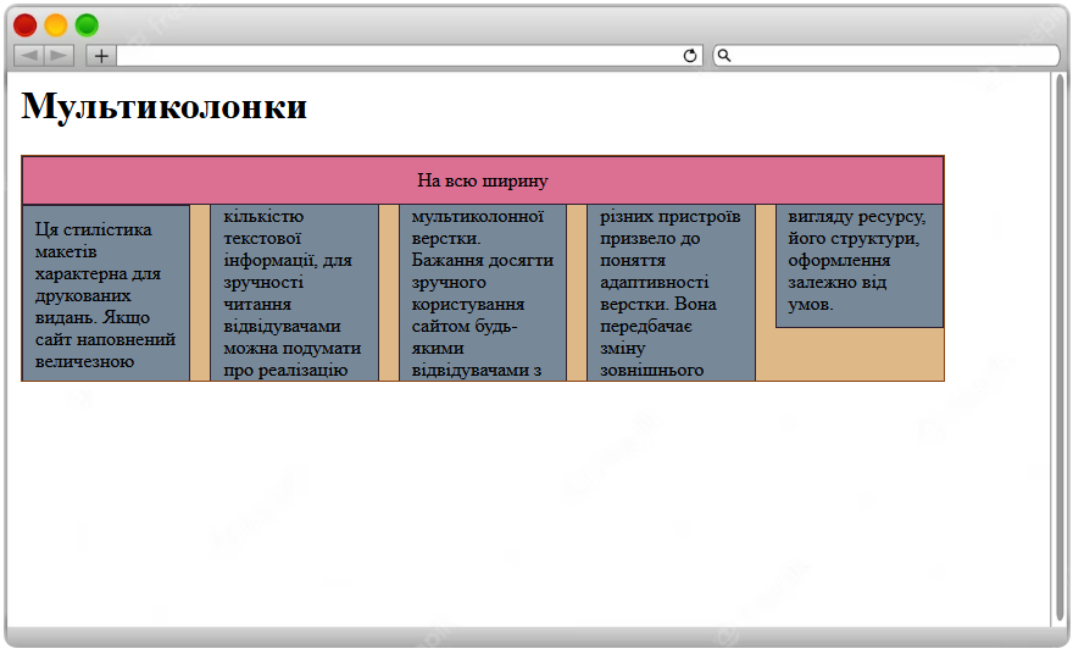


Рис.17. 5-ти колонковий макет

Адаптивний дизайн

Бажання досягти зручного користування сайтом будь-якими відвідувачами з різних пристроїв призвело до поняття адаптивності верстки. Вона передбачає зміну зовнішнього вигляду ресурсу, його структури, оформлення залежно від умов.

Для цього використовують так звані медіа-запити. Вони дозволяють враховувати тип пристрою, роздільну здатність екрану, орієнтацію.

Приклад - HTML

```
<h1>Тестуємо дозволи</h1>
<section>
</section>
```

Приклад - CSS

```
section {
      width: 90vw;
      height: 50vh;
}
@ media screen and (min-width: 200px){
      section {
         background-color: burlywood;
      }
}
@ media screen and (min-width: 700px){
      section {
         background-color: mediumaquamarine;
      }
}
```

```
@ media screen and (min-width: 1500px){
      section {
         background-color: olivedrab;
      }
}
```

Під час зміни розмірів активної області монітора змінюватиметься колір фону у тегу <section>.

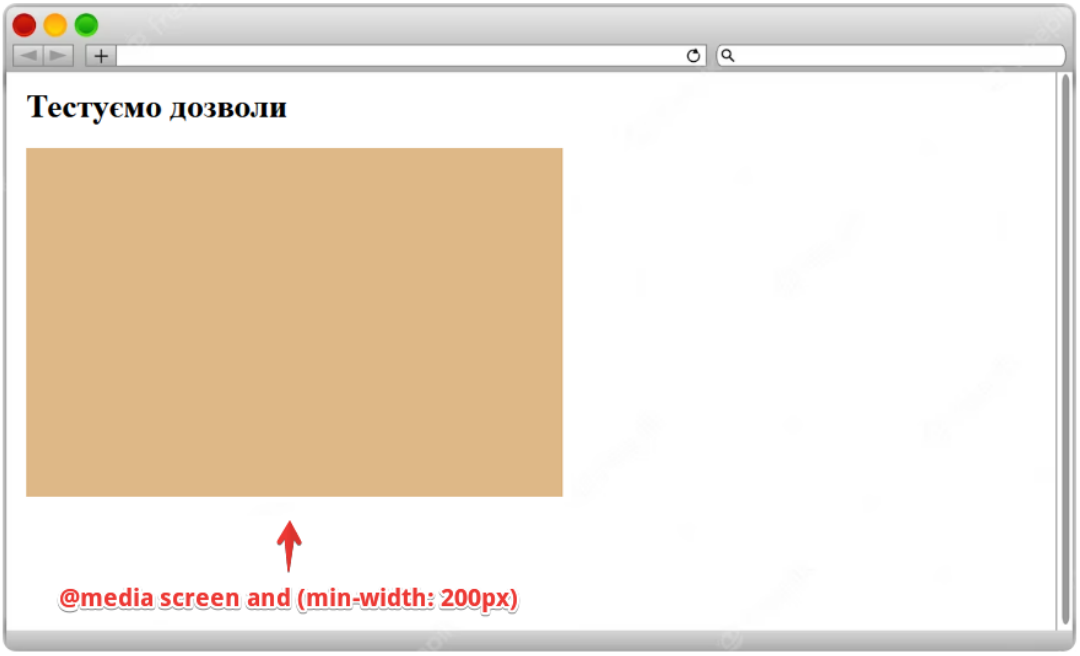


Рис.18. Коричневий колір фону при роздільній здатності > 200px, але менше 700px

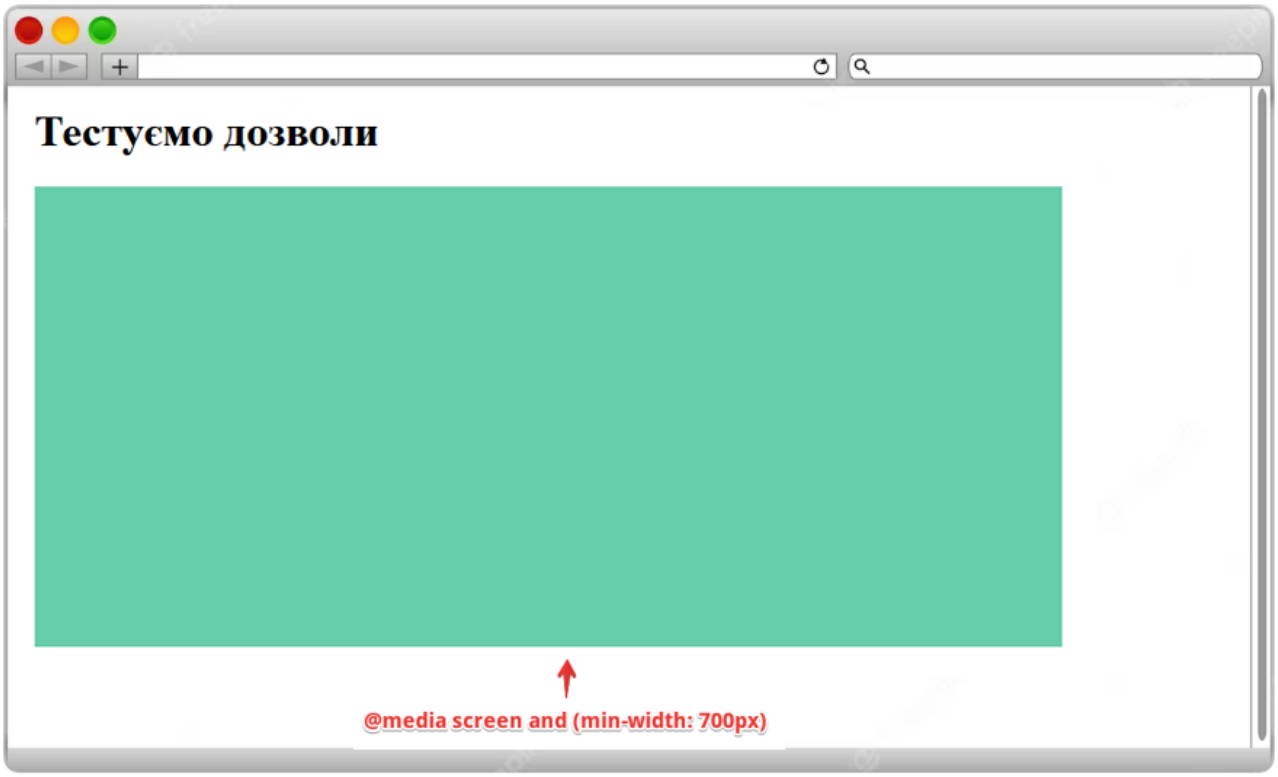


Рис.19. Фон стає при роздільній здатності > 700px, але <1500px

Подальші кроки

Вдосконалення навичок верстки потребує постійної практики. Найкраще її отримати тренуваннями: берете будь-який сайт, що вам сподобався, і намагаєтеся повторити його макет, не заглядаючи в інструменти розробника.

Зараз популярний фреймворк Bootstrap, ознайомитися з ним також важливо, як і з більш складними темами: анімації, функції.

Запитання

1. Які параметри впливають на підсумковий розмір блоку за шириною та висотою?

Якщо відкрити панель інструментів розробника в будь-якому браузері і виділити елемент сайту, то отримаємо наочну схему формування розміру цього блоку.

Він складається з: - ширини і висоти (width і height)

- внутрішніх відступів (padding)
- товщини рамки (border-width)
- зовнішніх відступів (margin)

2. Озвучте основні плюси методології Atomic.

Головні плюси підходу:

- Універсальні імена (наприклад, клас.text-input явно говорить про спосіб застосування. До картинок його застосовувати було б нелогічним)
- мультиплікативність класів (бо кожен клас відповідає за одну властивість, в більшості випадків, їх легко комбінувати, добиваючись множинного ефекту.font2rem red – два класи, один з яких задає розмір шрифту, а другий – робить текст червоним.Можливо використовувати як окремо, так і спільно)
- наочність імен (height100vh – клас, який сам за себе говорить: він робить висоту блоку, що дорівнює всій висоті вікна браузера)

3. У чому відмінність позиціонування sticky від fixed ?

При прокручуванні сторінки елемент з позиціонуванням fixed взагалі ніколи не рухається, залишаючись на тому самому місці. Завдання ж властивості sticky у міру вертикального прокручування поміщає елемент ближче до краю екрана, після чого він також стає нерухомим.

Завдання

Завдання 1

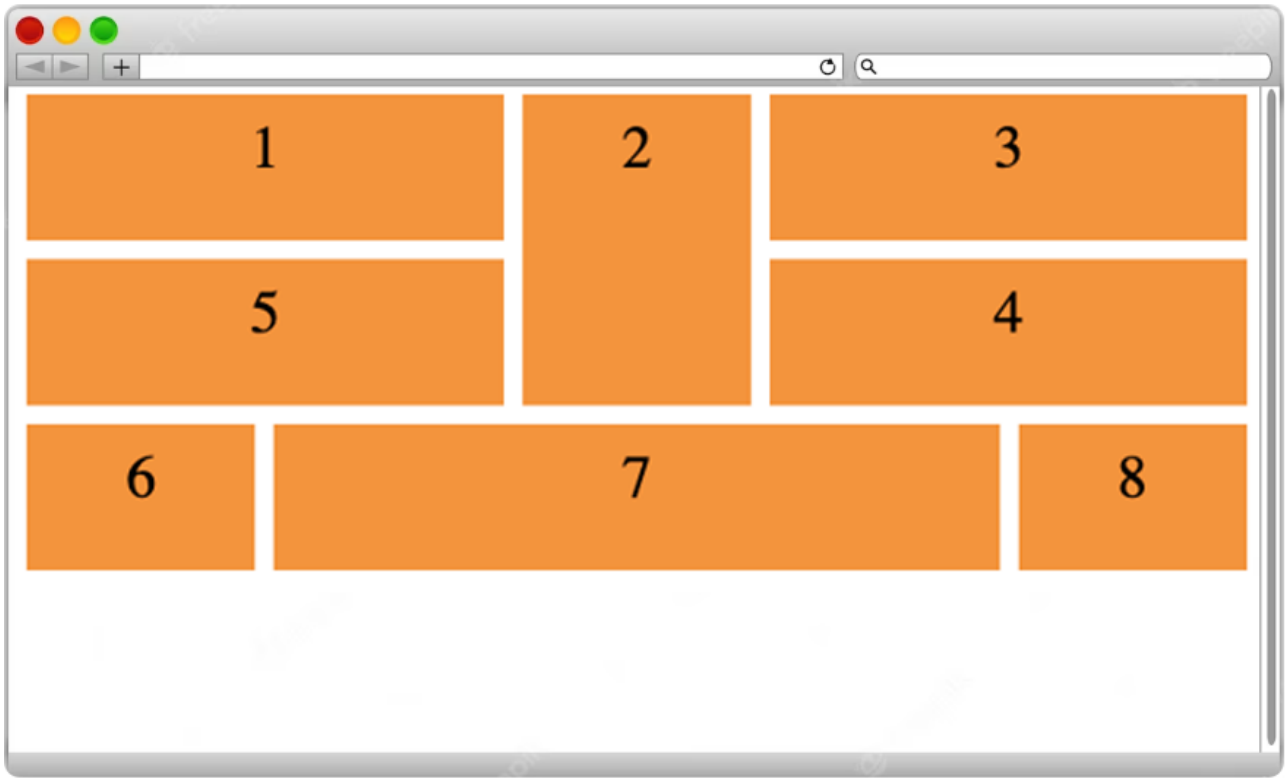
За допомогою HTML та CSS створіть список наступної структури:

c. Елемент

d. Елемент. Елемент == Пункт == Пункт == Пункт f. Елементg. ЕлементI. Поделемент II. Поделемент III. Поделементh. Елемент

Завдання 2

Створіть сітковий макет, що має структуру, показану на малюнку нижче.



Рішення

Завдання 1

Маємо 6 елементів на верхньому рівні нумерованого списку, у пункті e знаходиться ненумерований список, а у пункті g – нумерований перелік.

Рішення - HTML

```
<ol start="3">
      <li>Елемент</li>
      <li>Елемент</li>
      <li>Елемент
            <ul>
                  <li>Пункт</li>
                  <li>Пункт</li>
                  <li>Пункт</li>
            </ul>
      </li>
      <li>Елемент</li>
      <li>Елемент
            <ol>
                  <li>Пункт</li>
                  <li>Пункт</li>
```

```
                <li>Пункт</li>
            </ol>
        </li>
        <li>Елемент</li>
</ol>
```

Рішення - CSS

```
ol {
    list-style-type: lower-latin;
    list-style-position: inside;
}
ul {
    list-style-type: "==";
    list-style-position: inside;
}
li ol {
    list-style-type: upper-roman;
}
```

Завдання 2

Виходячи з макета, ми бачимо наступне: між осередками є відступи, деякі з них об'єднані. Усього виходить 5 колонок та 3 ряди. Як блоки можна взяти будь-які елементи.

Рішення - HTML

```
<div>
   <section>1</section>
   <section>2</section>
   <section>3</section>
   <section>4</section>
   <section>5</section>
   <section>6</section>
   <section>7</section>
   <section>8</section>
</div>
```

Рішення - CSS

```
div {
    display: grid;
    grid-template-columns: repeat(5, 1fr);
    grid-template-rows: repeat(3, 1fr);
    grid-column-gap: 10px;
    grid-row-gap: 10px;
    grid-auto-rows: 400px;
    font-size: 2rem;
```

```
}
section {
    background-color: rgb(255, 148, 41);
    min-height: 15vh;
    padding: 10px;
    text-align: center;
}

div section:first-child { grid-area: 1 / 1 / 2 / 3; }
div section:nth-child(2) { grid-area: 1 / 3 / 3 / 4; }
div section:nth-child(3) { grid-area: 1 / 4 / 2 / 6; }
div section:nth-child(4) { grid-area: 2 / 4 / 3 / 6; }
div section:nth-child(5) { grid-area: 2 / 1 / 3 / 3; }
div section:nth-child(6) { grid-area: 3 / 1 / 4 / 2; }
div section:nth-child(7) { grid-area: 3 / 2 / 4 / 5; }
div section:last-child { grid-area: 3 / 5 / 4 / 6; }
```

Джерела інформації

1. CSS. Стилі коду та побудова макетів https://smartiqa.ru/courses/web/lesson-6-css