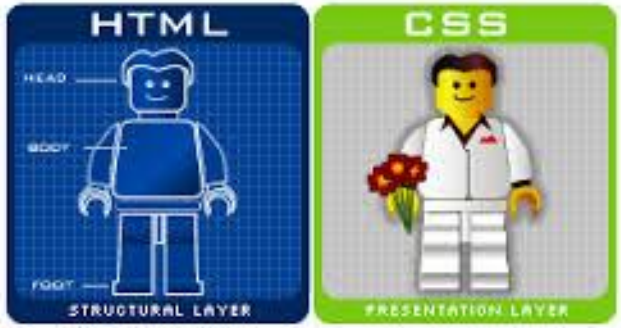
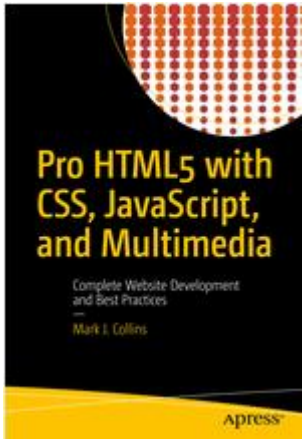
Курс “Основи web-програмування”
частина 2

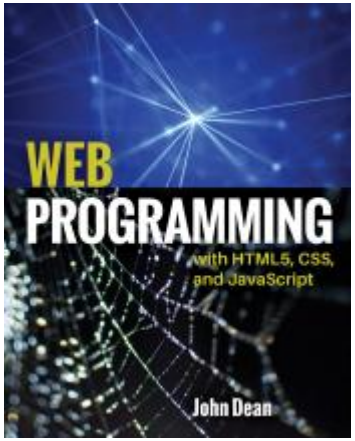


Мета – здобуття теоретичних знань і практичних навичок з основ web-технологій.

Література

1. Jörg Krause **Introducing Web Development** – Apress, 2016
2. Jennifer Robbins **Learning Web Design** Fifth Edition– O'Reilly Media, 2018. 790 p.
3. John Dean **Web Programming with HTML5, CSS and JavaScript** - Jones & Bartlett Learning, 2019. 678 p.
4. Mark J. Collins **Pro HTML5 with CSS, JavaScript, and Multimedia** - Apress, 2017. 560 p.





Що таке CSS ?

CSS (англ. *Cascading Style Sheets, -* ***Каскадні таблиці стилей***) - спеціальна мова опису сторінок, написаних мовою розмітки HTML

CSS видає браузеру інструкції про те, як вивести певний елемент

Приклад

```
p { font-family: Verdana, sans-serif; }

h2 { font-size: 110%; color: red; background: white; }

.note { color: red; background: yellow; font-weight: bold; }
```

CSS надає можливість розділення змісту сторінки (даних) та його візуальної презентації.

Корисні посилання

- W3 Schools (http://www.w3schools.com/)
- **Ресурси для розробників** (https://developer.mozilla.org/uk/)
- Генератор шаблонів сайтів - csstemplater.com
- Інструмент вибору кольору https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Colors/Color_picker_tool
- GoogleFonts https://fonts.google.com/
- HTML5 BOOK https://html5book.ru/
- **ТАБЛИЦЯ БЕЗПЕЧНИХ КОЛЬОРІВ** http://getcolorcode.com/ua/colors/websafe
- Free Image Placeholder Service https://placehold.it
- Lorem ipsum generator online http://uk.lorem-ipsum.info/_latin

- CSS ZEN GARDEN http://www.csszengarden.com/
- A Complete Guide to Flexbox https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- **Flexbox Froggy** http://flexboxfroggy.com/#ru

CSS

Після створення HTML-вмісту ми використовуємо CSS для визначення правил стилю для форматування вмісту. В цьому розділі ми розглянемо:

- **Selectors** - кожне правило CSS включає селектор, який визначає елементи, до яких правило слід застосувати.
- **Layout and positioning** – тут розглянено, як елементи розташовані відносно інших елементів та різні методи, доступні для контролю їх розташування.
- **Text** - тут розглянено, як вибрати шрифт та характеристики шрифту, вирівнювання тексту та пробіли, а також численні спеціальні ефекти, включаючи тіні та прикраси.
- **Borders and background**
- **Tables**
- **Flex**
- **Animation**

Джерела CSS стилів

CSS при відображенні сторінки може бути вибрана з різних джерел:

- **External style sheet - зовнішніх таблиць стилів,** тобто окремого файлу *.css*, на який робиться посилання в документі;
- **Embedded styles - вбудованих стилів** - блоків CSS всередині самого HTML-документа (тег style);
- **Inline styles - inline-стилів,** коли в HTML-документі інформація стилю для окремого елемента вказується у його атрибуті **style.**

Пріоритети

- Стандарт CSS визначає пріоритети, за якими застосовуються правила стилю, якщо для якогось елемента підходять кілька правил одночасно.
- Це називається «каскадом», в якому для правил визначаються пріоритети (переваги), що робить результати передбачуваними.
- В наведеному вище списку **стилі розміщені в порядку зростання пріоритетів**. (самі пріоритетні – inlyne-стилі)

Inline styles

- Додає стилі **до конкретного тегу** у файлі HTML

Example

```
<h1 style="color:red; font-family: sans-sarif">IU</h1>
```

Embedded styles

- Стиль застосовується **до всього HTML-файлу**
- Використовуйте його, коли вам потрібно змінити всі екземпляри певного елемента (наприклад, h1) на веб-сторінці

```
<head>
<title>Embedded Example</title>
<style>
        h1 {color:red; font-family:sans-serif}
</style>
</head>
```

External style sheet

- Файл styles.css:

  h1 {color:red; font-family:sans-serif;}

- Сторінка index.html:

  <head>

  <title>Getting Started</title>

  <link href="styles.css" rel="stylesheet" type="text/css" />

  </head>

  . . . . . .

Приклад

- Файл **external.css** : h1 { color: red }
- Файл index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>…</title>
  <style>
   @import url(external.css);  /* set to red first */
    h1 { color: purple;}  /* overridden by purple */
  </style>
</head>
<body>
  <h1 style="color: blue">Heading</h1>  /* blue comes last and
    wins */
  …
</body>
</html>
```

CSS Selectors

- Таблиці стилів складаються з набору правил. Кожне правило складається з одного або декількох **селекторів** і **блоку визначення.**
- **Блок визначення** містить набір правил відображення документа і записується в фігурних дужках у вигляді однієї чи кількох пар виду *"властивість : значення"*, які відокремлюються одна від одної крапкою з комою (;). Після останньої властивості крапка з комою необов'язкова.

Огляд селекторів

Селектор вибере нуль або більше елементів із документа HTML на основі певного аспекту елементів. Існує шість типів селекторів, кожен з яких використовує різну інформацію в елементі для здійснення вибору. Ці типи селекторів:

- **Element** selectors (селектори тегів)
- **Class** selectors
- **ID** selectors
- **Attribute** selectors
- **Pseudo-class** selectors
- **Pseudo-elements**

CSS UNITS OF MEASUREMENT

- Час ознайомитись з **одиницями вимірювання**, що використовуються в CSS.
- Ви будете використовувати їх **для встановлення розміру шрифту, ширини та висоти елементів, полів, відступів** тощо.
- Розглянемо повний список "CSS Units“
- CSS3 забезпечує різноманітні одиниці вимірювання. Вони поділяються на дві широкі категорії: абсолютну (**absolute**) та відносну (**relative**).

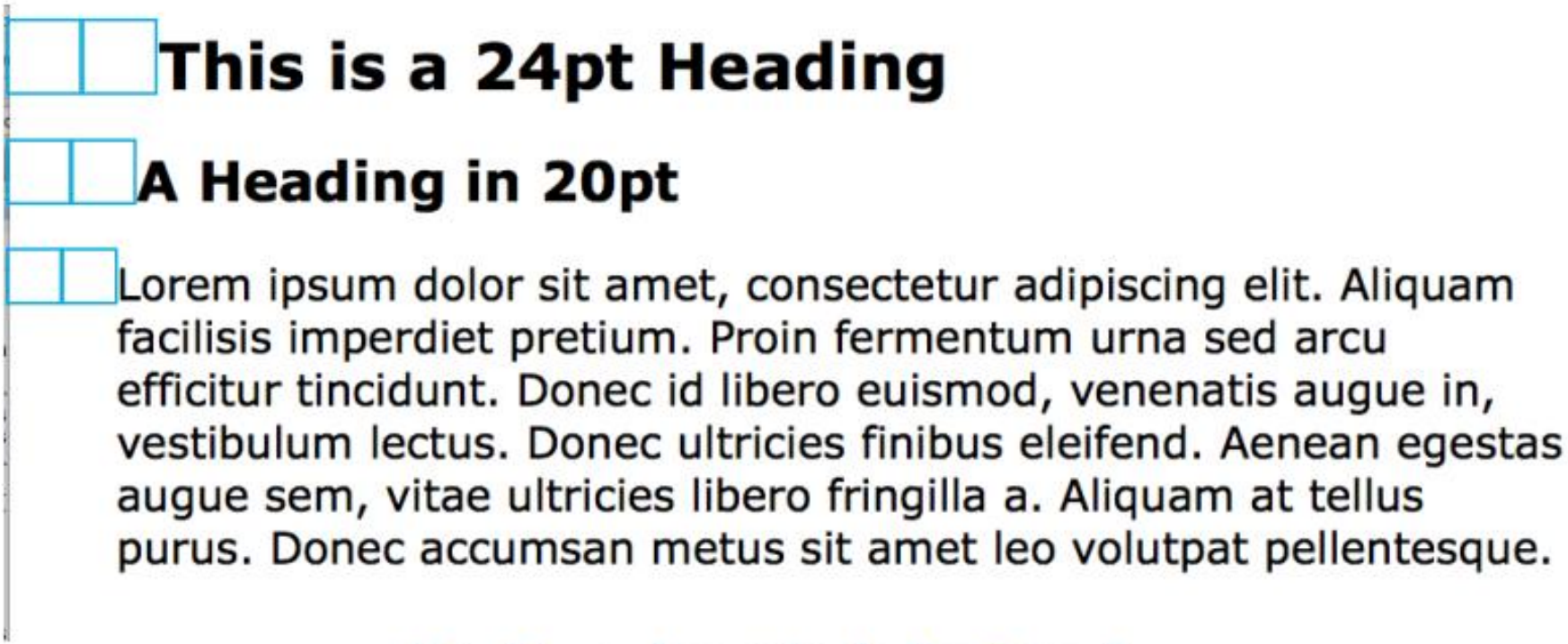
Absolute units

- **px** pixel, defined as equal to 1/96 of an inch in CSS3.
- **in** inches.
- **mm** millimeters.
- **cm** centimeters.
- **q** ¼ millimeter.
- **pt** points (1/72 inch). Points are a unit commonly used in print design.
- **pc** picas (1 pica = 12 points or 1/6 inch). Points are a unit commonly used in print design.

Relative units

- **em** a unit of measurement equal to the current font size.
- **ex** x-height, approximately the height of a lowercase “x” in the font.
- **rem** root em, equal to the em size of the root element (html).
- **ch** zero width, equal to the width of a zero (0) in the current font and size.
- **vw** viewport width unit, equal to 1/100 of the current viewport (browser window) width.
- **vh** viewport height unit, equal to 1/100 of the current viewport height.
- **vmin** viewport minimum unit, equal to the value of vw or vh, whichever is smaller.
- **vmax** viewport maximum unit, equal to the value of vw or vh, whichever is larger.

Приклад 1

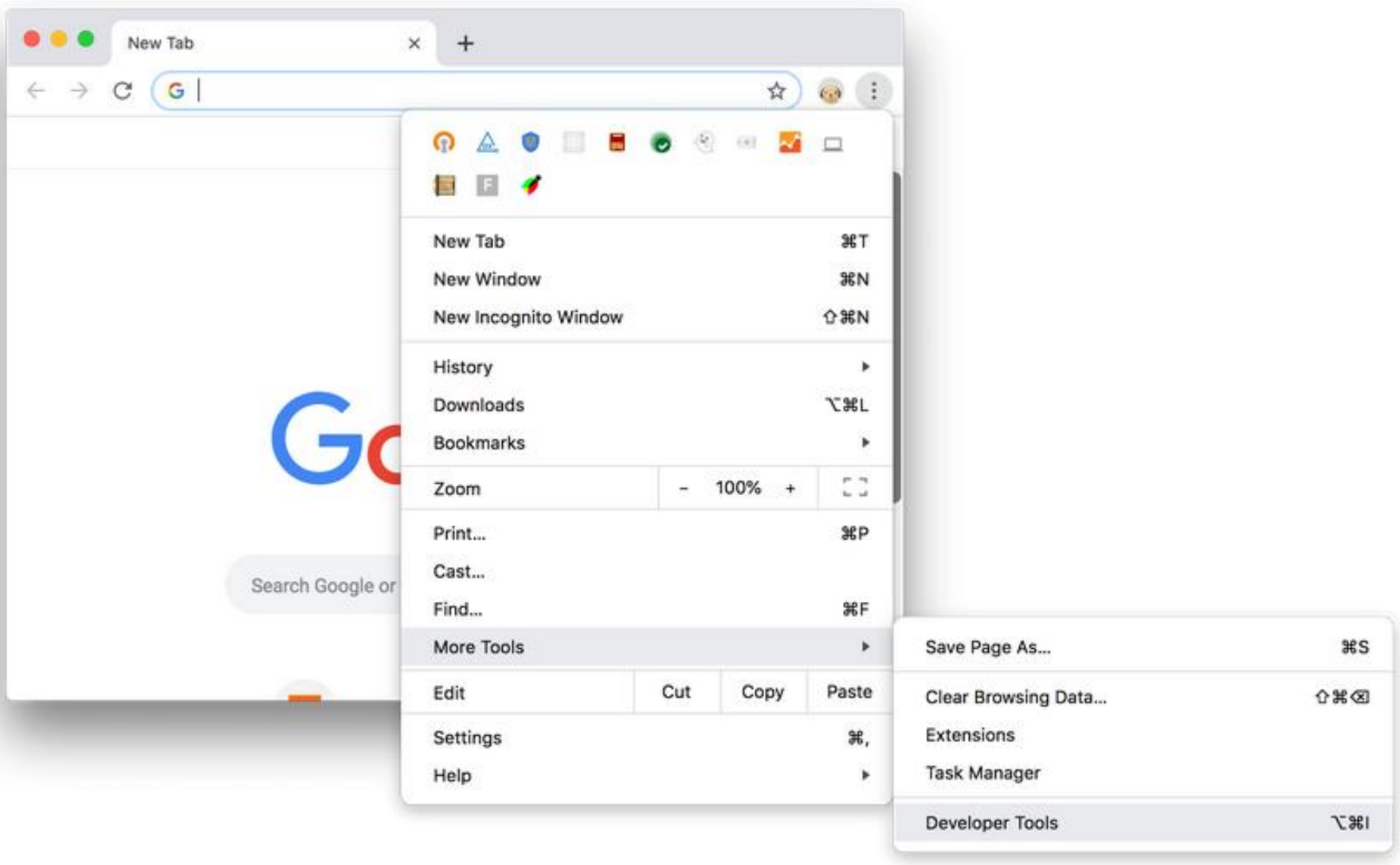


- Дивись більш детально про units :
  https://www.w3.org/TR/css3-values/

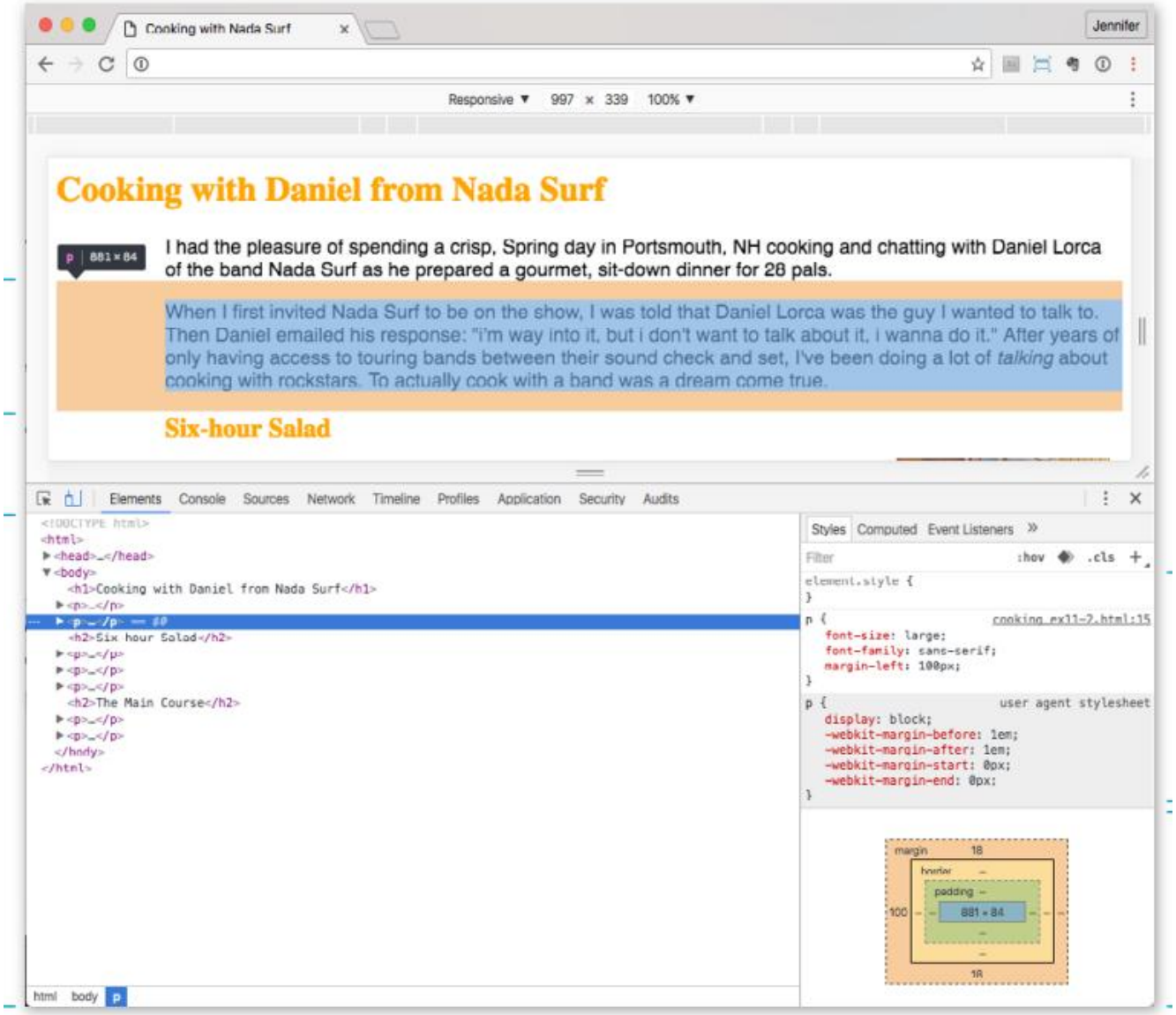
Приклад 2


```
6   * set the second div's height to 200 pixels
7   * set the third div's margin to 1 em
8   * set the fourth div's font-size 2 ems
9
10  -->
11
12  <html>
13  <head>
14      <title>Quiz - Units in CSS</title>
15      <style>
16        .first {
17          width: 100px;
18        }
19        .second {
20          height: 200px;
21        }
22        .third {
23          margin: 1em;
24        }
25        .fourth {
26          font-size: 2em;
27        }
28      </style>
```

Open DevTools from Chrome's main menu



Developer Tools panel



- Файл cooking.html (з каталогу Приклади)

1) Element Selectors

```
h1 {
    border-width: 1px;
    border-style: solid;
    border-color: black;
    font-size: 72px;
    font-family: Verdana;
    text-align: center;
}
```

```
header,
aside,
footer {
    background-color: yellow;
}
```

2) Class Selectors

```
<!DOCTYPE html>
<html>
<head>
<style>
.intro {
  background-color: yellow;
}
</style>
</head>
<body>

<h1>Welcome to My Homepage</h1>

<div class="intro">
  <p>My name is Donald.</p>
  <p>I live in Duckburg.</p>
</div>

<p>My best friend is Mickey.</p>

<p class="intro">My best friend is Mickey.</p>

</body>
</html>
```

Тест

- Який із перелічених нижче елементів HTML відповідає заданому виразу CSS?

```
.right {
  text-align: right;
}
```

- ☐ `<div class="right"></div>`
- ☐ `<a href="#" class="leftright"></a>`
- ☐ `<button id="right"></button>`
- ☐ `<p class="highlight module right"></p>`

3) ID Selectors

ID selector працює так само, як і class selector, за винятком того, що він використовує атрибут id замість class, і ви префіксуєте його хеш-символом (#) таким чином:

```
#id-name { property:value; }
```

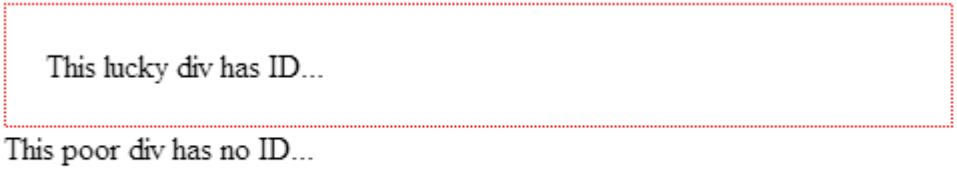
ID selector вказує один елемент на основі його **унікального ідентифікатора**, тому за визначенням стиль не буде використаний повторно.

Краще визначити стилі на основі елементів або класів, щоб подібні елементи можна було стилізувати однаково.

ID selector повинні використовуватися ощадливо і лише для унікальних ситуацій, коли стиль не потрібно використовувати повторно.

Приклади

```
<!DOCTYPE html>
<title>Example</title>
<style>
  div#css-section {
    border:1px dotted red;
    padding: 20px;
    }
</style>
<div id="css-section">
  This lucky div has ID...
</div>
<div>
  This poor div has no ID...
</div>
```




```
<style>
 #send {
   color: red;
 }
</style>
. . . .
<button id="send">Send</button>
```

4) ID Selectors

- Ви також можете використовувати селектор ідентифікаторів **як частину контекстного селектора**.
- У наступному прикладі стиль застосовується лише до елементів, які відображаються всередині елементу, ідентифікованого як "resources".
- Таким чином, посилання у елементі, названому "resources", відображаються інакше, ніж всі інші посилання на сторінці.

```
#resources a { text-decoration: none; }
```

5) Attribute Selectors

- Селектори атрибутів надають вам велику гнучкість, дозволяючи вибирати елементи на основі будь-якого з атрибутів елемента.
- Їх вказують як [атрибут = значення], подібно до цього:

```
[class="book"] {
    background-color:yellow;
}
```

Це функціонально еквівалентно використанню селектора класу .book; проте, селектор атрибутів дозволяє виконувати відповідність, використовуючи лише частини значення атрибута. Для цього прикріпіть знак рівності (=) з одною із наступних дій:

- **~** (наприклад, [class ~= "book"]): Значення атрибута повинно містити слово, вказане значенням селектора (наприклад, class = "some book titles"). Саме так працює селектор класів.
- **|** (наприклад, [class |= "book"]): Значення атрибута повинно починатися зі слова, яке відповідає значенню селектора (наприклад, class = ""book titles")
- **^** = (наприклад, [class ^= "book"]): значення атрибута повинно починатися зі значення селектора (наприклад, class = "books")
- **$** (наприклад, [class $= "book"]): значення атрибута має закінчуватися значенням селектора (наприклад, class = "checkbook")
- ***** (наприклад, [class *= "book"]): значення атрибута повинно містити значення селектора (наприклад, class = "overbooked")

- Ви можете **вказати атрибут без значення**, який поверне всі елементи, які мають атрибут. Хорошим прикладом цього є селектор [href], який вибере всі елементи, що мають атрибут href, незалежно від його значення.
- Ви також можете включити element selector перед селектором атрибутів для подальшого обмеження вибраних елементів. Наприклад, це поверне всі елементи img, атрибут src починається з https:

```
img[src^="https"] {
    color:blue;
}
```

6) Pseudo-Class Selectors

- Псевдокласи схожі на звичайні атрибути класу, за винятком того, що вони **додаються браузером автоматично**, а не встановлюються в розмітці HTML.
- Псевдокласи мають префікс двокрапка (:).
- Багато з них динамічно застосовуються залежно від стану елементів. Розглянемо, наприклад, гіперпосилання. Якщо по посиланню вже переходили, зазвичай посилання відображається з іншим кольором. Це досягається за допомогою такого правила CSS, яке змінить колір усіх елементів, які мають :visited псевдоклас.

```
:visited {
    color: blue;
}
```

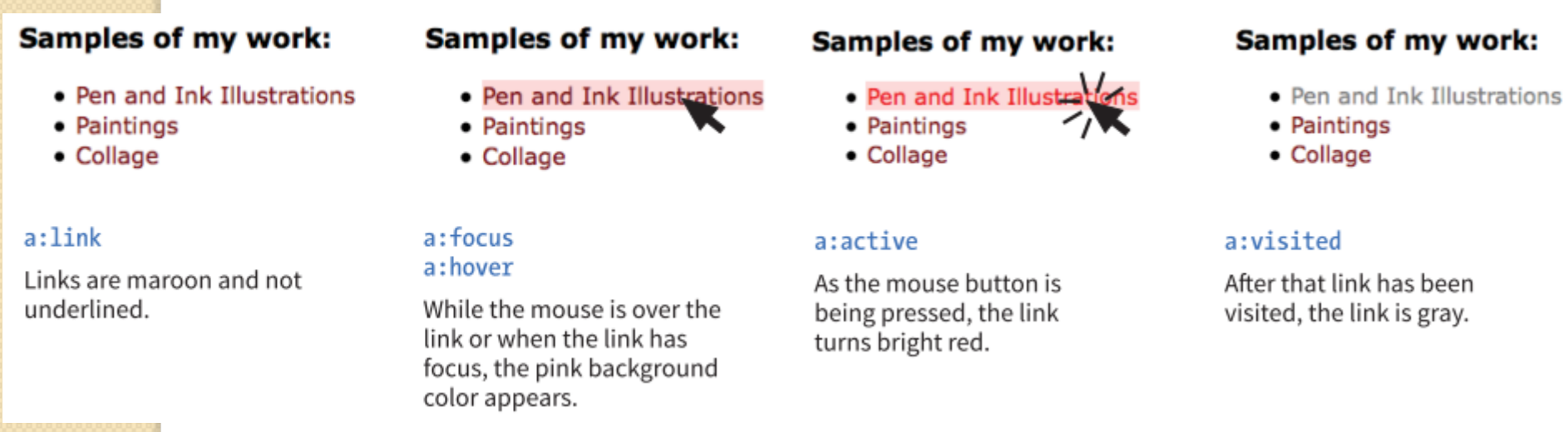
Link Pseudo-Classes

```
a:link {
  color: maroon;
}
a:visited {
  color: gray;
}
input:focus { background-
    color: yellow; }
a:hover {
  color: maroon;
background-color: #ffd9d9;
}
a:active {
  color: red;
  background-color: #ffd9d9;
}
```

The required order for pseudo-classes is:

- :link
- :visited
- :focus
- :hover
- :active

```
a { text-decoration: none; }  /* turns underlines off for all links */
a:link { color: maroon; }
a:visited { color: gray; }
a:focus { color: maroon; background-color: #ffd9d9; }
a:hover { color: maroon; background-color: #ffd9d9; }
a:active { color: red; background-color: #ffd9d9; }
```



Деякі псевдокласи

- :active - Вибирає посилання, щойно натиснуте
- :checked - Вибирає елементи, які відмічені (стосується прапорців)
- :focus - Вибирає елемент, на який в даний час є фокус
- :hover - Вибирає елемент, над яким зараз наведена миша
- :link - Вибирає всі непроглянуті посилання
- :visited - Вибирає всі відвідані посилання
- :default - Вибирає на формі елемент за замовчуванням, як правило, кнопку submit
- :first-child - Вибирає елементи, які є першою дитиною його безпосереднього батька
- :last-child - Вибирає елементи, які є останньою дитиною в її батьків

- :nth-child(n) - Вибирає елементи, які є n-ою дитиною в її батьків.

Приклад.

```
<ol>
   <li>c</li>
   <li>Beta</li>
   <li>Gamma</li>
   <li>Delta</li>
   <li>Epsilon</li>
   <li>Zeta</li>
   <li>Eta</li>
   <li>Theta</li>
   <li>Iota</li>
   <li>Kappa</li>
</ol>
```

1) **ol :nth-child(2) { color: orange; }**

“Beta” will be orange

2) **ol :nth-child(2n) { color: orange; }**

“Beta,” “Delta,” “Zeta,” “Theta” and “Kappa” will be orange

3) **ol :nth-child(even) { color: orange; }**

Всі непарні стануть orange

4) **ol :nth-child(2n+6) { color: orange; }**

“Zeta,” “Theta” and “Kappa” will be orange

Pseudo-Elements

- У той час як псевдокласи забезпечують механізм вибору елементів, псевдоелементи фактично повертають нові віртуальні елементи, які ви можете стилізувати, фактично не входячи в DOM. Це або порожні елементи, або частина існуючого елемента.
- Псевдоелементи починаються з подвійної двокрапки (: :), щоб відрізнити їх від псевдокласів. Це доступні псевдоелементи:
- ::after - це створює порожній елемент відразу після вибраних елементів
- ::before - це створює порожній елемент безпосередньо перед обраними елементами
- ::first-letter: Вибирає перший символ кожного обраного елемента
- ::first-line:
- ::selection : Повертає частину елемента, обраного користувачем

- Ви можете додати ::before або ::afterкваліфікатори до селектора, щоб вставити вміст у документ до або після вибраних елементів. Використовуйте ключове слово **content**: для вказівки вмісту та включення будь-яких бажаних команд стилю (стиль стосується лише вставленого вмісту). Наприклад, щоб додати "Важливо!" Перед кожним тегом p, який одразу слідує за тегом заголовка, використовуйте наступне правило.

```
header+p:before {
  content:"Important! ";
  font-weight:bold;
  color:red;
}
```

Універсальний селектор

- У версії CSS2 був введений універсальний селектор елемента (*), який відповідає будь-якому елементу (іншими словами, є чимось на зразок джокера).
- Правило стилів

  * {color: gray; }

зробить відображення кожного елемента в документі сірим кольором.

- Цей селектор також корисний як контекстуальний, як показано в цьому прикладі, який вибирає **всі елементи** в розділі «intro»:

  #intro * { color: gray; }

Використання комбінаторів

- Різні типи селекторів, які ми розглянули, також можна комбінувати для виконання більш складних виборів.

1) Поєднання елементів і класових селекторів

- Ви можете комбінувати селектори елементів і класів. Наприклад, тут будуть обрані всі елементи абзаців, які мають представлений клас:

```
p.featured {
}
```

- Ви також можете комбінувати декілька селекторів класів. Вони будуть оброблені з логічним оператором AND. Наприклад, це вибирає всі елементи абзаців, які мають як featured, так і new класи:

```
p.featured.new {
}
```

2) Pseudo-Selectors

- Селектори псевдокласів часто поєднуються із селектором елементів, наприклад:

```
a:visited {
}
```

Це вибирає всі теги посилань, які вже відвідали. У цьому випадку псевдоклас додатково уточнює селектор елемента.

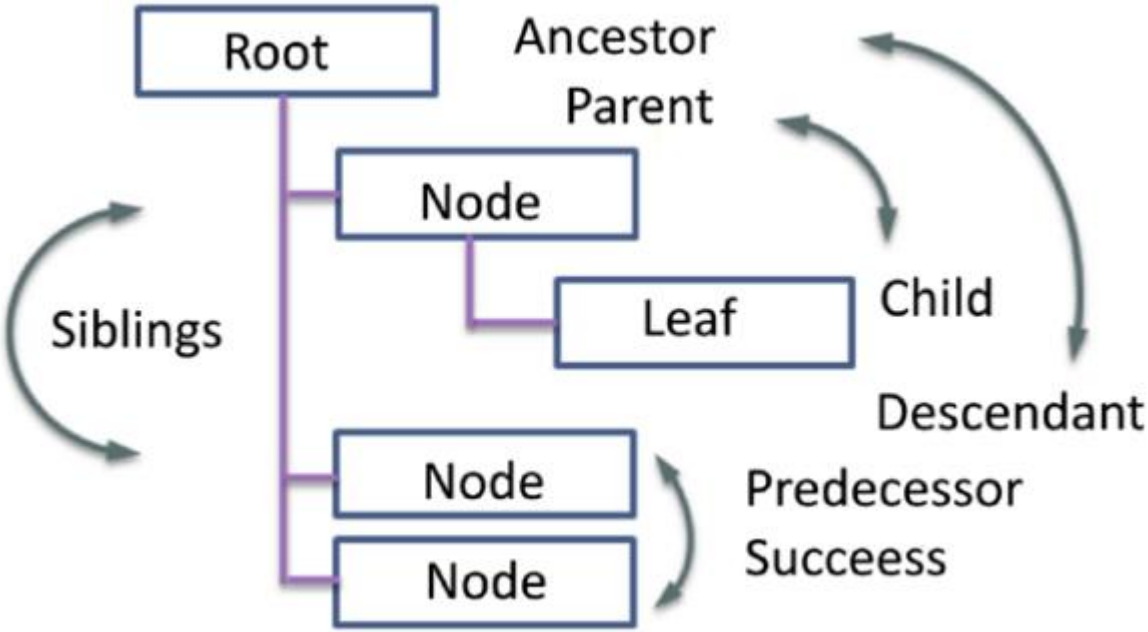
Їх також можна комбінувати з селектором класів, так:

```
.featured:focus {
}
```

Однак псевдокласи також можуть стояти самостійно.

```
:default {
}
```

Elements in the hierarchy of a HTML site



- Ancestor – предок
- Descendant – нащадок
- Siblings – братні елементи

3) Оператори комбінаторів

Ви можете комбінувати селектори, щоб вказати певні ієрархії елементів. Комбінуючи елементи з одним із наступних комбінаторів, ви можете створити більш складний селектор:

- **Group** , (for example p, h1): Логічний оператор OR вибирає всі p елементи, а також усі елементи h1.
- **Descendant space** (Простір нащадків) (for example, header p): Вибирає другий елемент, коли він знаходиться всередині першого елемента. Наприклад, ви хочете вибрати усі елементи **p** що всередині елемента header.
- **Child** (Дитина) **>** (for example header>p): Вибирає другий елемент, коли перший елемент є безпосереднім батьківським. Селектор header> p повертає всі p елементи, безпосереднім батьківським елементом яких є header.

- Adjacent Sibling (Суміжні брати) + (for example header+p): Вибирає другий елемент, коли перший елемент є попереднім братом другого елемента.
- Follows (іде слідом) ~ (for example p~header): Вибирає другий елемент, коли той йде за першим елементом (не обов'язково відразу).

Щоб проілюструвати останні два правила, якщо ваш документ виглядає наступним чином, селектор **h1 + p** не поверне жоден елемент, але обидва **h2 + p** і **h1 ~ p** повернуть елемент p:

```
<h1>Some header</h1>
<h2>Some sub-header</h2>
<p>Some text</p>
```

The Not Selector

- Ви також можете додати префікс :not до будь-якого селектора. (Означає: не повертати всі вибрані елементи). Наприклад, це вибирає всі елементи в тілі, крім елементів заголовка:

```
body:not(header) {
color:purple;
}
```

Пріоритетність стильових правил
(Мейер Э., Эстелл У. - CSS. Полный справочник. , 2019, с. 125)

Приклад. Яке правило більш пріоритетне?

```
hl {color: red;}
body hl {color: green;}

h2.grape {color: purple;}
h2 {color: silver;}
```

Пріоритетність селектора визначається його компонентами і представляється чотиризначним виразом формату 0, 0, 0, 0. На пріоритетність селектора впливають такі фактори.

- **Кожен ідентифікатор** (атрибут id) збільшує пріоритетність на величину 0, 1, 0, 0.
- **Селектори класів**, псевдокласів і атрибутів додають до пріоритетності величину 0, 0, 1, 0.

- За кожен **селектор елементів** і псевдоелементів нараховується пріоритетність 0, 0, 0, 1.
- **Комбінатори** і універсальні селектори на значення пріоритетності не впливають.

Приклади.

```
hl {color: red;}          /* пріоритетність = 0,0,0,1 */
p em {color: purple;} /* пріоритетність = 0,0,0,2 */
.grape {color: purple;} /* пріоритетність = 0,0,1,0 */
*.bright {color: yellow;} /* пріоритетність = 0,0,1,0 */
p.bright em.dark {color: maroon;} /*пріоритетність = 0,0,2,2 */
#id216 {color: blue;} /* пріоритетність = 0,1,0,0 */
div#sidebar *[href] {color: silver;}/*пріоритетність = 0,1,1,1*/
```

Якщо застосувати **друге** і **п'яте** правило з наведених вище прикладів до елементу **em**, то він отримає темно-бордовий колір (maroon), оскільки у п'ятого правила значення пріоритетності більше, ніж у другого.

Важливість стилів

- Окремі стилі настільки важливі, що повинні застосовуватися першочергово, незалежно від їх базового рівня пріоритетності. В CSS такі стилі називаються **важливими** і позначаються за допомогою спеціального ключового слова !important, яке додається в кінець оголошення, якраз перед крапкою з комою:

  p.dark {color: #333 !important; background: white;)

Успадкування

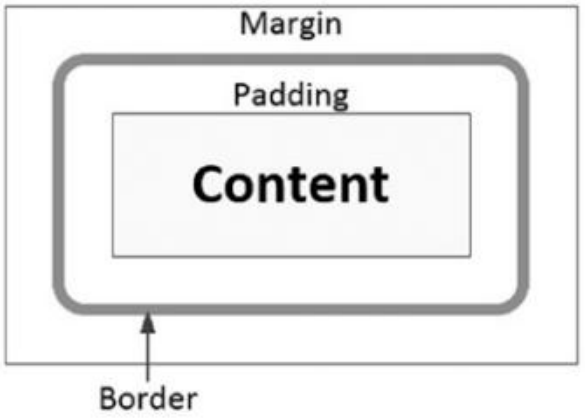
На порядок застосування стильових правил в документі впливає не тільки їх пріоритетність, а й ще одна ключова концепція CSS: успадкування. Спадкування є механізм передачі стильового форматування від батьківського елемента до його нащадкам. Наприклад, встановлюючи колір елементів h1, ви змінюєте колір тексту не тільки заголовків першого рівня, а й усіх його дочірніх елементів.

```
hl {color: gray;}
<hl>Meerkat <em>Central</em></hl>
```

Box Model (Блокова модель)

- HTML підтримує два види форм опису елементів - елементи потоку та елементи блоку.
- **Елементи потоку** вбудовуються в текст. Ці елементи не мають розмірів, таких як ширина і висота, оскільки вони залежать від навколишніх елементів.
- Однак **елементи блоку мають розміри** і при необхідності зміщують будь-які сусідні елементи.
- Кожен блоковий елемент займає певний простір, який залежить від вмісту цього елемента. Крім того, на це впливають такі фактори, як padding та margin.

Padding (поля) - це простір між вмістом та межею (border) елемента.
Margin (відступи)- це простір між border та сусідніми елементами. Це проілюстровано на рисунку.



Components of the box model

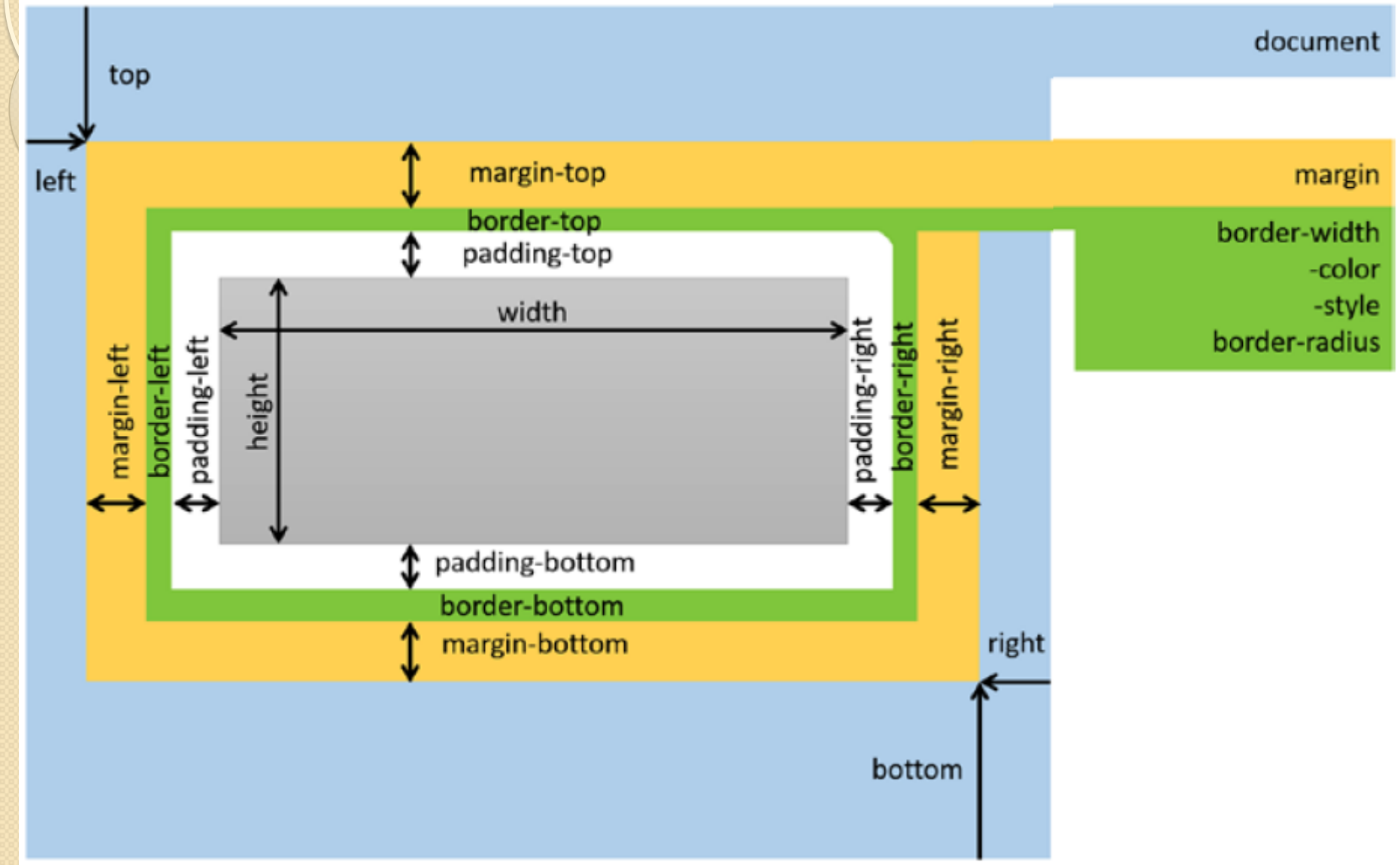


TABLE 3-1 The `padding` Shorthand Property

Syntax	Description
`padding: value1;`	Applies *value1* to all four sides
`padding: value1 value2;`	Applies *value1* to the top and bottom and *value2* to the right and left
`padding: value1 value2 value3;`	Applies *value1* to the top, *value2* to the right and left, and *value3* to the bottom
`padding: value1 value2 value3 value4;`	Applies *value1* to the top, *value2* to the right, *value3* to the bottom, and *value4* to the left

Collapsing margins

```
nav {
    margin-top: .5rem;
    padding: .75rem;
    border: 1px solid black;
}
<header>
    <img src="/images/notw.png" alt="News of the Word logo">
    <h1>News of the Word</h1>
    <h3>Language news you won’t find anywhere else (for good
reason!)</h3>
</header>
<nav>
    <a href="#">Home</a>
    <a href="#">What's New</a>
    <a href="#">What's Old</a>
    <a href="#">What's What</a>
</nav>
```

Додавання

```
header {
    margin-bottom: .5rem;
}
```

Нічого не змінює!

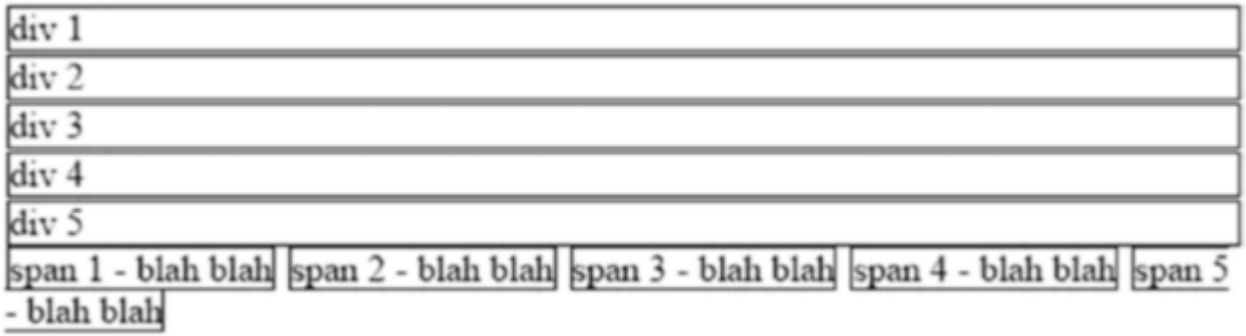
Positioning Content

- HTML-документ можна вважати рядом boxes (блоків). Документ складають **структурні елементи**, які визначають великі поля, такі як header, section та елементи article. Вони визначають структуру вашого документа.
- У межах них ви розміщуєте **більш дрібні елементи**, такі як абзаци та елементи зображення.
- А в межах елементів абзацу ви включаєте безліч маленьких блоків із використанням phrasing елементів, таких як елементи strong, emphasis та span.
- В цьому розділі ми **розглянемо кілька понять**, які потрібно зрозуміти, щоб ефективно викласти вміст HTML-документа.

1) Display

- Найбільш фундаментальним **атрибутом** щодо компонування в CSS є атрибут **display**. Є два основні варіанти: block та inline.
- Є й інші підтримувані значення, які ми розглянемо пізніше. Але поки зосередимося на цих двох .
- **block** - елемент використовує всю ширину свого батьківського елемента. Додаткові блокові елементи укладаються вертикально, трохи нижче попереднього елемента.
- **inline** - елемент розташовується горизонтально. Кожен inline елемент розміщується праворуч від попередніх елементів. Якщо для елемента недостатньо місця, деякий або весь вміст елемента переноситься на наступний рядок.
- **none** - елемент взагалі не відображається. Вміст не тільки приховано, але він не займає місця на екрані.

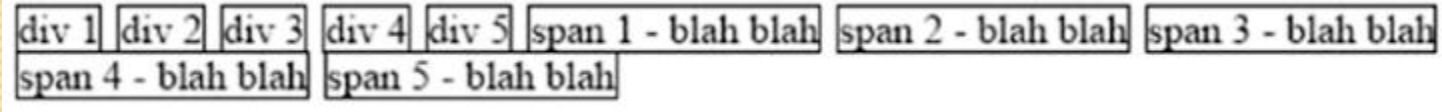
Приклад 1




```
<div>div 1</div>
<div>div 2</div>
<div>div 3</div>
<div>div 4</div>
<div>div 5</div>
<span>span 1 - blah blah</span>
<span>span 2 - blah blah</span>
<span>span 3 - blah blah</span>
<span>span 4 - blah blah</span>
<span>span 5 - blah blah</span>

div, span {
  border: 1px solid black;
  margin: 1px;
}
```

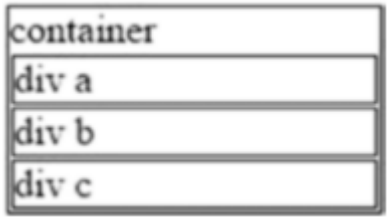
Що додати, щоб виглядало так?



Приклад 2

```
<div class="container">container
<div>div a</div>
<div>div b</div>
<div>div c</div>
</div>

.container {
   width: 150px;
}
```



Defining Sizes

Існує чимало способів визначення розміру елемента, і кожен із цих методів має наслідки.

- Absolute Size

У попередньому прикладі ми встановили атрибут width, надавши йому абсолютну ширину 150 пікселів. Ви також можете встановити атрибут height.

```
.container {
width: 150px;
height: 100px;
}
<div class="container">container
<div>div a</div>
<div>div b</div>
<div>div c</div>
<div>div d</div>
<div>div e</div>
</div>
<div>More content</div>
```

- Relative Size

```
.container {
    width: 70%;
}
```

- Setting Maximum Values

Якщо ви в основному маєте справу з текстом, визначити ширину контейнера корисно для обмеження області, яка буде містити текст. Текст буде обгорнуто в міру необхідності, щоб вмістится у вказаній області. Однак якщо доступного простору менше, ніж вміст, то вміст перекриє цю область, як було ілюстровано раніше.

Просте вдосконалення для вирішення цього питання - використання атрибута **max-width** замість ширини. Замініть обидві декларації в правилі CSS на максимальну ширину:

```
.container {
   max-width: 300px;
}
```

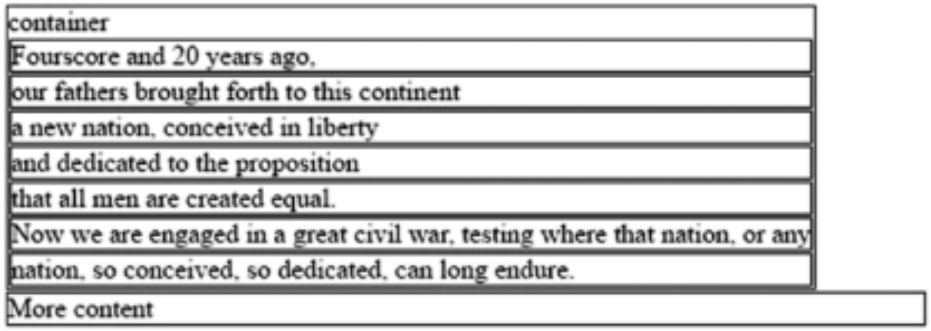
• Content-Based

Є два параметри, за допомогою яких можна визначити ширину елемента на основі його вмісту:

- min-content - використовує найменшу можливу область, яка відповідає вмісту після використання всіх можливостей упаковки.
- max-content - використовує найменший простір, необхідний без упаковки будь-якого вмісту.

Приклад

```
.container {
    width: -moz-max-content;
    height: -moz-max-content;
    width: max-content;
    height: max-content;
}
```



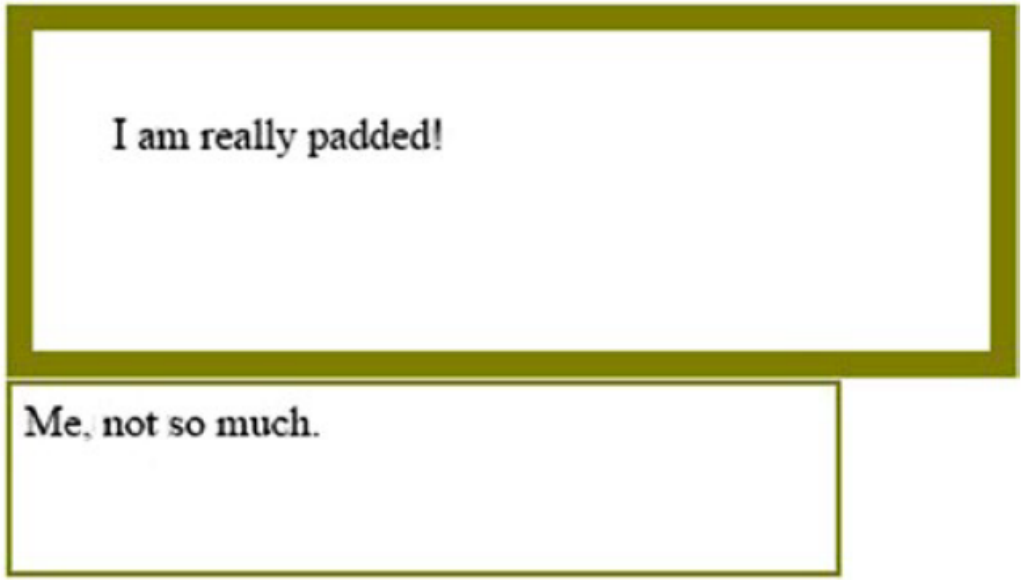
Box Sizing

Проблема.

```
<section>
    <article class="bigBorder">I am really padded!</article>
    <article class="smallBorder">Me, not so much.</article>
</section>
```

CSS:

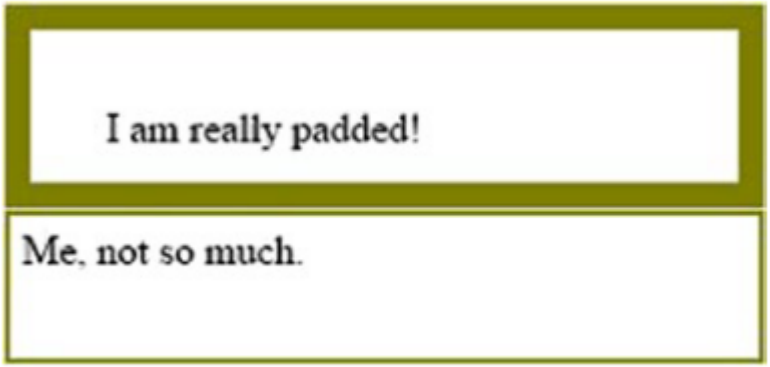
```
article {
width: 300px;
height: 60px;
margin: 1px;
}
.bigBorder {
padding: 30px;
border:  10px solid olive;
}
.smallBorder {
padding: 5px;
border:  2px solid olive;
}
```



Рішення: новий атрибут box-sizing

- Цей атрибут визначає, як застосовуються атрибути висоти та ширини.
- Значення за замовчуванням, content-box, вказує, що розмір застосовується лише до фактичного вмісту елемента.
- Крім того, ви можете встановити його в border-box, і атрибут розміру буде застосовано після додавання padding та border.
- Ми можемо виправити попередній приклад, додавши таке правило CSS:

```
* {
  box-sizing: border-box;
}
```

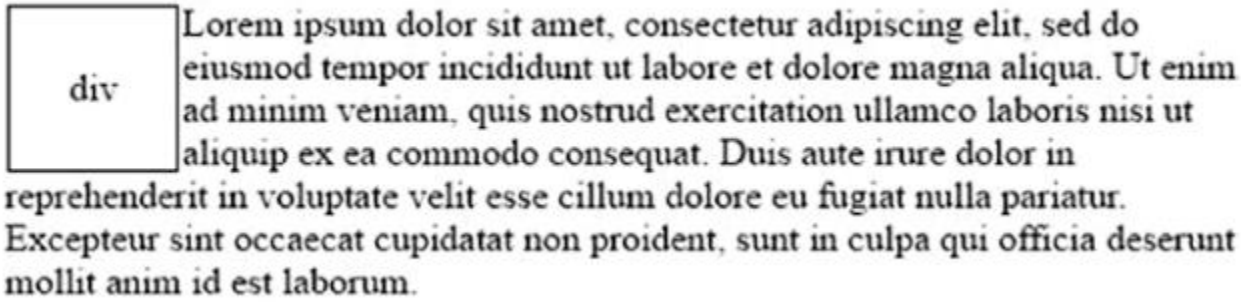


2) Float (обтікання)

- Атрибут float використовується для того, щоб змусити вирівняти вміст блоку або по **лівому**, або по **правому** краю елемента, що його містить.

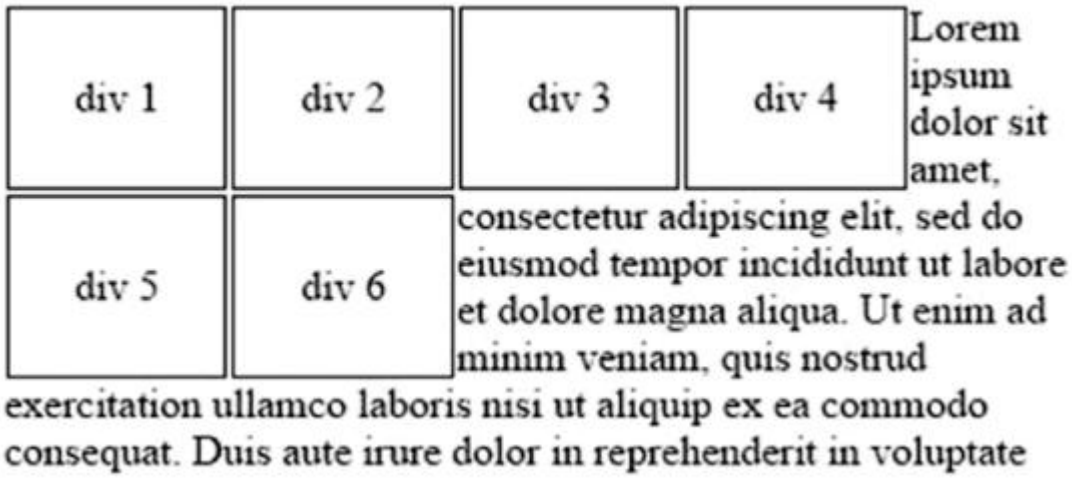
```
<header>
<div>div</div>
<p> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
   ...</p>
</header>
```

```
header>div {
  padding: 25px;
  float: left;
}
```



Пам'ятайте, що вміст блоку зазвичай **займає всю ширину контейнера**. Встановлення атрибута **float** на елементі дозволяє наступному вмісту розміщуватись так, **ніби він був inline**. Додаткові елементи розміщуються праворуч від плаваючого вмісту.

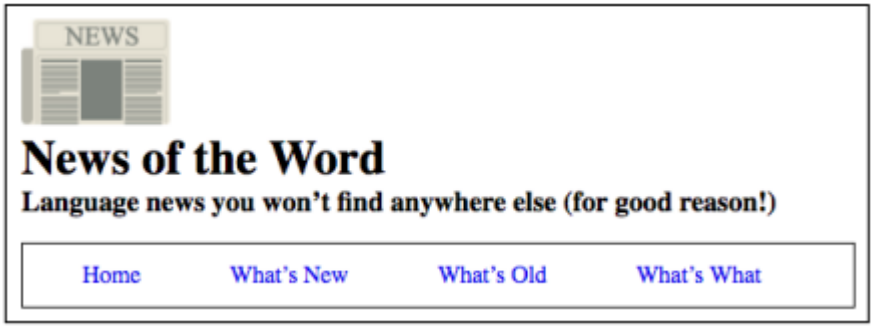
```
<header>
<div>div 1</div>
<div>div 2</div>
<div>div 3</div>
<div>div 4</div>
<div>div 5</div>
<div>div 6</div>
 <p>
 Lorem ipsum dolor sit amet,
 consectetur adipiscing elit,
 sed do eiusmod ...
 </p>
 </header>
```

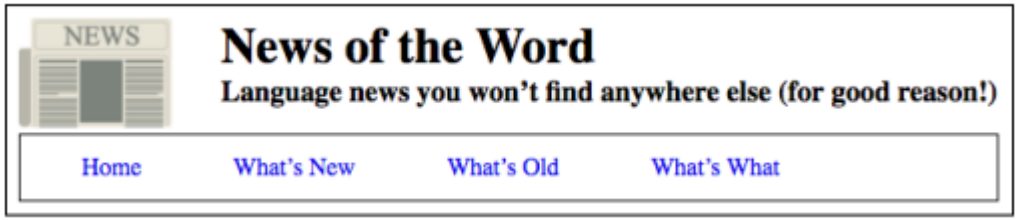


Приклад

```
<header>
    <img src="/images/notw.png" alt="News of the Word logo">
    <h1>News of the Word</h1>
    <h3>Language news you won't find anywhere else (for good
reason!)</h3>
</header>
<nav>
    <a href="#">Home</a>
    <a href="#">What's New</a>
    <a href="#">What's Old</a>
    <a href="#">What's What</a>
</nav>

header img {
    float: left;
    margin-right: 2em;
}
```





2b) Clearing Floats

- Типова поведінка для неплаваючих речей - це обгортання всього, що плаває, що часто саме те, що ви хочете.
- Однак будуть випадки, коли ви хочете уникнути того, щоб елемент обертався навколо ваших floats. Наприклад, розглянемо наступний код.

```
<header>
    <h1>Can't You Read the Sign?</h1>
</header>
<nav>
    <a href="/">Home</a>
    <a href="semantics.html">Signs</a>
    <a href="contact.html">Contact Us</a>
    <a href="about.html">Suggest a Sign</a>
</nav>
<article>
    <img src="/images/keep-off-the-grass.jpg">
</article>
<footer>
    &copy; Can't You Read?, Inc.
</footer>
```

With the <img> tag floated to the left.

- Щоб виправити ситуацію треба додати властивість clear:

```
element {
    clear: left|right|both|none;
}
```

Використовуйте **clear: left**, щоб очистити всі left-floated елементи, **clear: right**, щоб очистити всі right-floated елементи або **clear: both**, щоб очистити все.

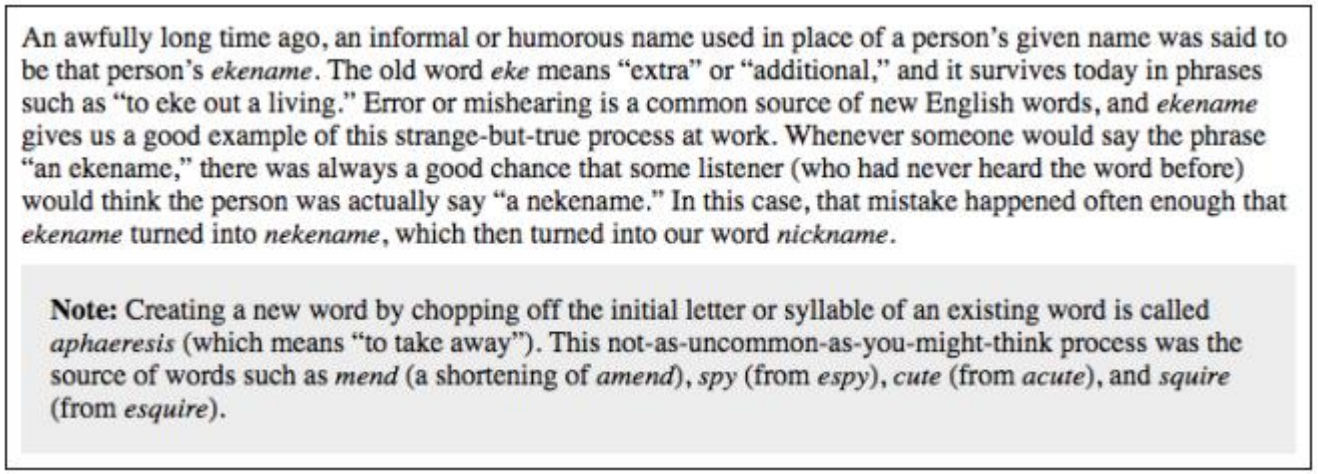
Для нашого прикладу:

```
footer {
    clear: left;
}
```

Collapsing containers

- Дивна поведінка CSS, мабуть, безмежна, і floats пропонують ще один приклад. Розглянемо наступний HTML та його результат на малюнку:

```
<article>
    <section>
        An awfully long time ago...
    </section>
    <aside>
        <b>Note:</b> Creating a new word by...
    </aside>
</article>
```

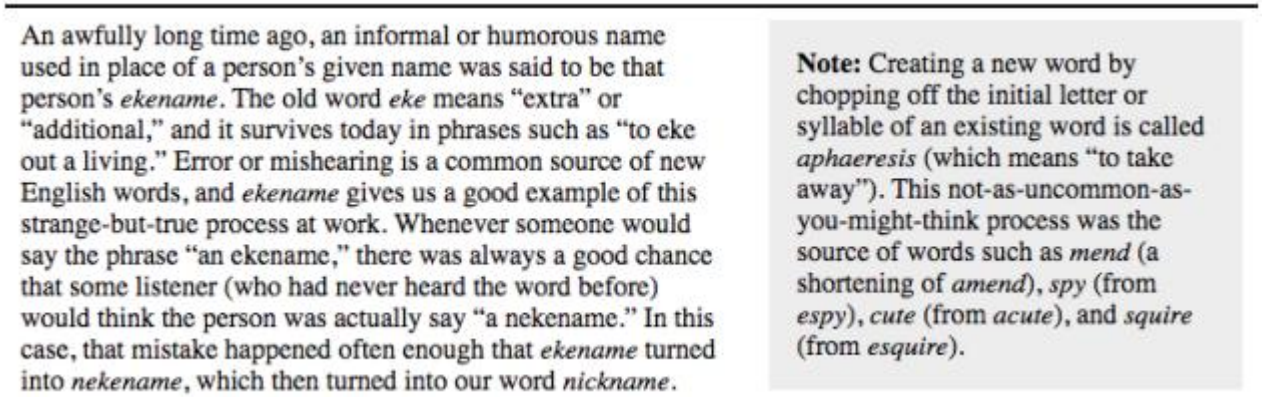


An awfully long time ago, an informal or humorous name used in place of a person's given name was said to be that person's *ekename*. The old word *eke* means "extra" or "additional," and it survives today in phrases such as "to eke out a living." Error or mishearing is a common source of new English words, and *ekename* gives us a good example of this strange-but-true process at work. Whenever someone would say the phrase "an ekename," there was always a good chance that some listener (who had never heard the word before) would think the person was actually say "a nekename." In this case, that mistake happened often enough that *ekename* turned into *nekename*, which then turned into our word *nickname*.

Note: Creating a new word by chopping off the initial letter or syllable of an existing word is called *aphaeresis* (which means "to take away"). This not-as-uncommon-as-you-might-think process was the source of words such as *mend* (a shortening of *amend*), *spy* (from *espy*), *cute* (from *acute*), and *squire* (from *esquire*).

- Зауважте, зокрема, що ми стилізували елемент статті рамкою.
- Тепер спробуємо покласти елементи section і aside поруч.

```
section {
    float: left;
    width: 25rem;
}
aside {
    float: right;
    width: 15rem;
}
```



An awfully long time ago, an informal or humorous name used in place of a person's given name was said to be that person's *ekename*. The old word *eke* means "extra" or "additional," and it survives today in phrases such as "to eke out a living." Error or mishearing is a common source of new English words, and *ekename* gives us a good example of this strange-but-true process at work. Whenever someone would say the phrase "an ekename," there was always a good chance that some listener (who had never heard the word before) would think the person was actually say "a nekename." In this case, that mistake happened often enough that *ekename* turned into *nekename*, which then turned into our word *nickname*.

Note: Creating a new word by chopping off the initial letter or syllable of an existing word is called *aphaeresis* (which means "to take away"). This not-as-uncommon-as-you-might-think process was the source of words such as *mend* (a shortening of *amend*), *spy* (from *espy*), *cute* (from *acute*), and *squire* (from *esquire*).

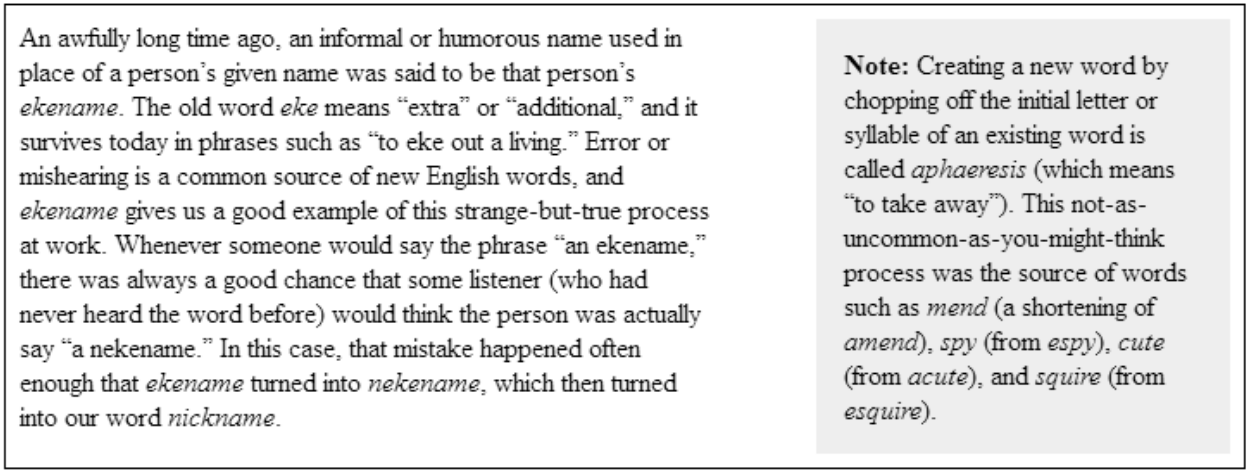
The article element has collapsed!

Що трапилось? Оскільки ми floated як section, так і aside елементи, браузер видалив їх із потоку сторінок, завдяки чому елемент статті поводився так, ніби він взагалі не мав вмісту. Результат? CSS bugaboo (страховище), відомий як **крах контейнера**.

- Щоб виправити це, ви повинні змусити батьківський контейнер очистити власних дітей

CSS:

```
.self-clear::after {
    content: "";
    display: block;
    clear: both;
}
```



An awfully long time ago, an informal or humorous name used in place of a person's given name was said to be that person's *ekename*. The old word *eke* means "extra" or "additional," and it survives today in phrases such as "to eke out a living." Error or mishearing is a common source of new English words, and *ekename* gives us a good example of this strange-but-true process at work. Whenever someone would say the phrase "an ekename," there was always a good chance that some listener (who had never heard the word before) would think the person was actually say "a nekename." In this case, that mistake happened often enough that *ekename* turned into *nekename*, which then turned into our word *nickname*.

Note: Creating a new word by chopping off the initial letter or syllable of an existing word is called *aphaeresis* (which means "to take away"). This not-as-uncommon-as-you-might-think process was the source of words such as *mend* (a shortening of *amend*), *spy* (from *espy*), *cute* (from *acute*), and *squire* (from *esquire*).

HTML:

```
<article class="self-clear">
```

3) Position

Способи позиціонування

Специфікація CSS визначає **п'ять способів позиціонування** контейнерів елементів, що встановлюються за допомогою властивості position.

```
element {
    position: static|relative|absolute|fixed;
}
```

- static: Розміщує елемент у його стандартному положенні у потоці сторінки; це значення за замовчуванням.
- relative - Положення елемента встановлюється щодо місця початкового його розташування.
- absolute - Елемент витягується із загального потоку елементів документа, а його положення визначається відносно положення блоку що його містить (або іноді більш раннього предка).
- fixed - елемент розташовується відносно viewport; його положення на екрані не змінюється, коли документ прокручується.

- Зміщення елемента (offsets) визначаються наступними властивостями CSS:

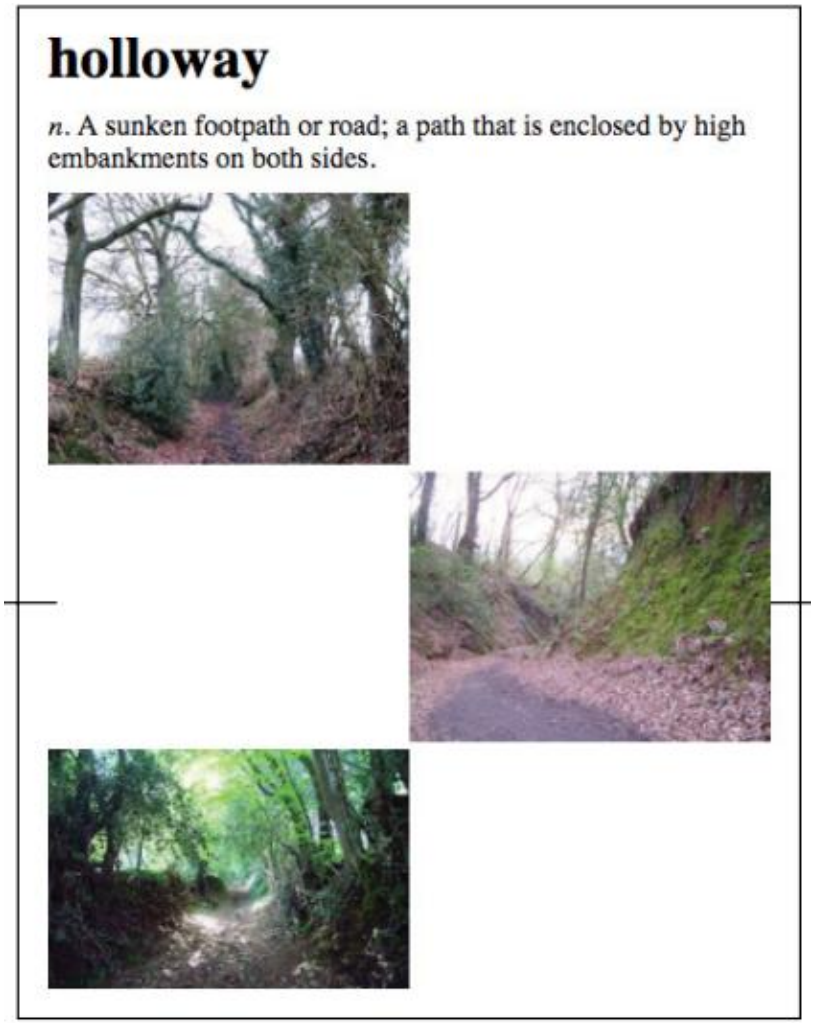
```
element {
    top: top-value;
    right: right-value;
    bottom: bottom-value;
    left: left-value;
}
```

Relative позиціонування

Приклад.

```
.offset-image {
    position: relative;
    left: 200px;
}
```

```
<h1>
   holloway
</h1>
<div>
   <i>n.</i> A sunken footpath or
   road; a path that is enclosed
by high embankments on both sides.
</div>
<img src="images/holloway1.jpg">
<img src="images/holloway2.jpg"
   class="offset-image">
<img src="images/holloway3.jpg">
```



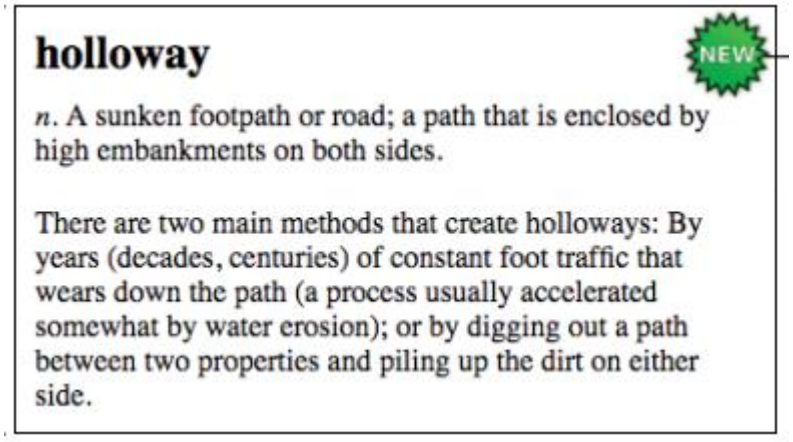
Absolute positioning

- Абсолютне позиціонування не тільки здвигає елемент із його положення за замовчуванням, але й видаляє елемент із потоку сторінки.
- Q.: Відносно якого елемента обчислюється здвиг (offset)?
- A.: Відносно найближчого елемента предка, який використовує нестатичне позиціонування.

Приклад.

```
section {
    position: relative;
    border: 1px double black;
}
img {
    position: absolute;
    top: 0;
    right: 0;
}
```

```
<section>
    <img src="images/new.png">
    <h1>
      holloway
    </h1>
    <div>
      <i>n.</i> A sunken footpath or road; a path that is
enclosed by high embankments on both sides.
    </div>
    <div>
      There are two main . . . </div>
</section>
```

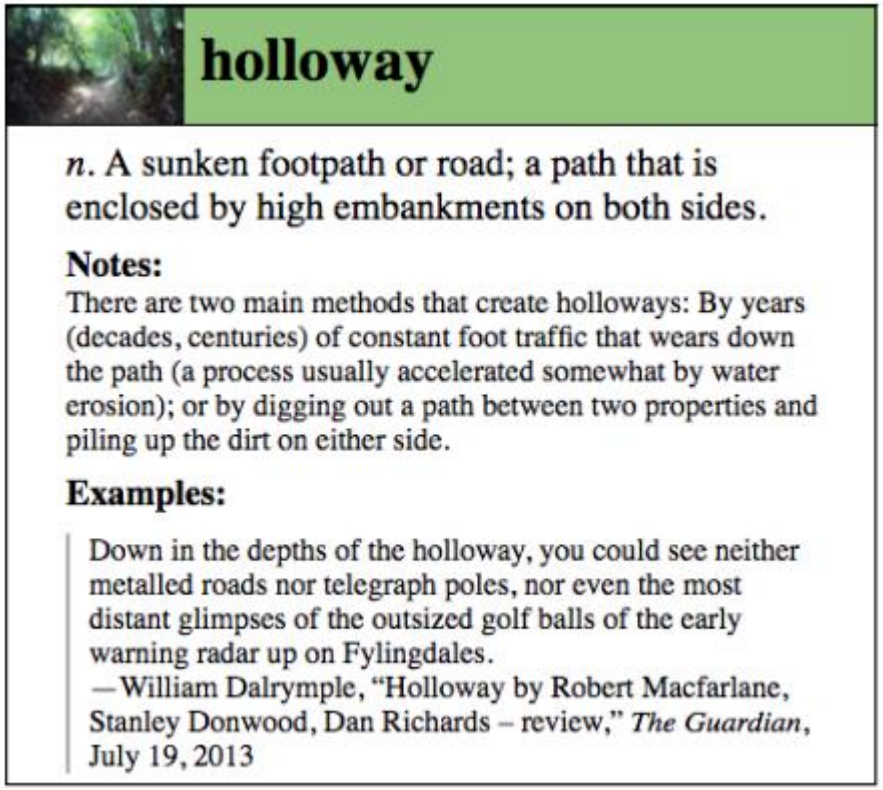


Fixed Positioning

- Фіксоване позиціонування ідентичне абсолютному позиціонуванню, за винятком того, що контекст позиціонування завжди є **viewport** (вікно перегляду).
- Вміст завжди залишатиметься на одному і тому ж місці на екрані, незалежно від того, як документ прокручується - горизонтально або вертикально.

```
header {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 64px;
    border: 1px double black;
    background-color: rgb(147, 196, 125);
}
main {
    margin-top: 64px;
}
```

```
<header>
    <img src="images/holloway3.jpg">
    <h1>
        holloway
    </h1>
</header>
<main>
...
</main>
```



holloway

n. A sunken footpath or road; a path that is enclosed by high embankments on both sides.

Notes:

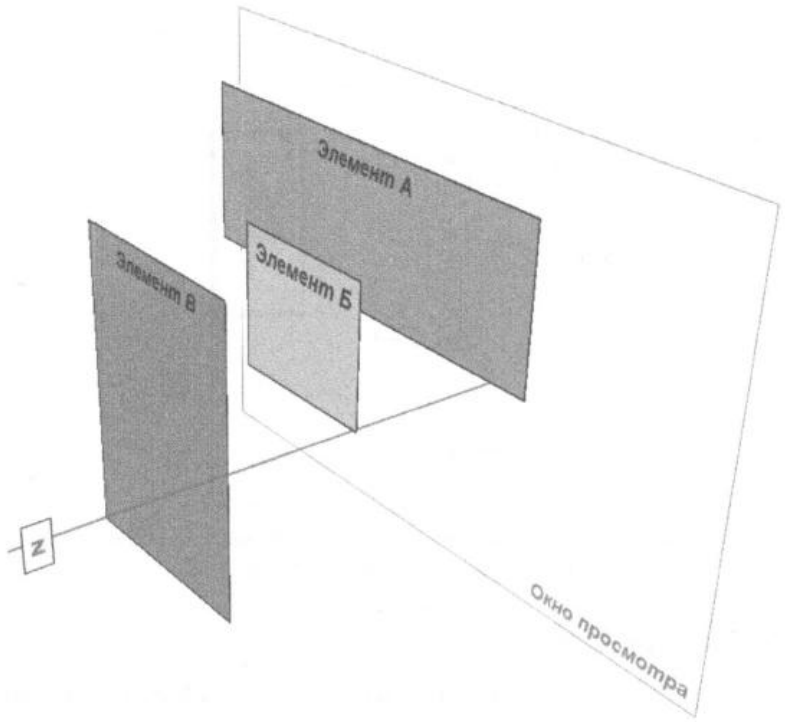
There are two main methods that create holloways: By years (decades, centuries) of constant foot traffic that wears down the path (a process usually accelerated somewhat by water erosion); or by digging out a path between two properties and piling up the dirt on either side.

Examples:

> Down in the depths of the holloway, you could see neither metalled roads nor telegraph poles, nor even the most distant glimpses of the outsized golf balls of the early warning radar up on Fylingdales.
> —William Dalrymple, "Holloway by Robert Macfarlane, Stanley Donwood, Dan Richards – review," *The Guardian*, July 19, 2013

Порядок накладення елементів. Z-index

- При абсолютному позиціонуванні елементів часто виникають ситуації, коли одне і те ж місце документа претендують зайняти відразу кілька елементів.
- В CSS порядок розташування елементів один над одним встановлюється властивістю **z-index**.
- z-index має значення за замовчуванням 0.



Приклад

```
<p id="A"><img src="A.gif" alt="A"></p>
<p id="B"><img src="B.gif" alt="B"></p>
<p id="C"><img src="C.gif" alt="C"></p>

#A {
  z-index: 100;
  position: absolute;
  top: 175px;
  left: 200px;
}
#B {
  z-index: 5;
  position: absolute;
  top: 275px;
  left: 100px;
}
#C {
  z-index: 1;
  position: absolute;
  top: 325px;
  left: 250px;
}
```

Що таке макет сторінки?

- Макет сторінки - це розташування елементів сторінки в зоні вмісту веб-переглядача.
- Макет сторінки виконує функцію креслення сторінки, як і будь-який хороший план, у макеті сторінки детально описано, як сторінка виглядає на двох рівнях:
- **Макрорівень**: стосується загального макету сторінки, який визначає, як основні розділи сторінки - header, nav, main, footer, тощо – працюють разом у цілому.
- **Мікрорівень**: стосується макету у межах розділу чи підрозділу сторінки. Наприклад, елемент заголовка сторінки може мати один макет, тоді як у розділі статті сторінки може бути інший.

Методи компонування (Page Layout)

CSS пропонує чотири основні методи компонування, кожен з яких можна застосувати як на макрорівні, так і на мікрорівні:

- Floats: Впорядковує елементи використовуючи властивість float.
- Inline blocks: Впорядковує елементи, стилізуючи їх як inline блоки.
- CSS Flexible Box (flexbox): Розташовує елементи вертикально або горизонтально в межах гнучких блоків.
- CSS Grid: Впорядковує елементи в структурі рядків і стовпців.

1) Розміщення елементів сторінки за допомогою Floats

Загальна процедура, якої ви повинні дотримуватися, йде приблизно так:

1. Працюйте вниз по сторінці, дозволяючи елементам сторінки викладатися за допомогою потоку сторінок за замовчуванням.
2. Коли ви перейдете до двох або більше елементів, які ви хочете бачити поруч, float їх left (зазвичай) або right.
3. Коли ви перейдете до наступного елемента, який повинен слідувати за потоком сторінки за замовчуванням, очистіть floats для цього елемента.
4. Повторюйте кроки від 1 до 3, поки не досягнете кінця сторінки.

Приклад 1

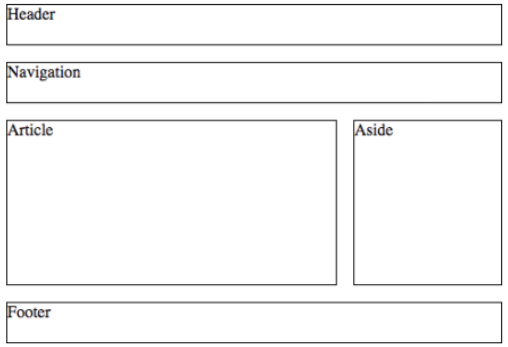
```
nav {
    height: 2.5rem;
    padding-top: .6rem;
    background-color: #ccc;
}
nav ul {
    list-style-type: none;
    padding-left: 1.75rem;
}
nav li {
    float: left;
    padding-right: 1.75rem;
}
main {
    clear: left;
    margin-top: 1rem;
}
```

```
<nav>
    <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Blog</a></li>
        <li><a href="#">Store</a></li>
        <li><a href="#">About</a></li>
        <li><a
    href="#">Contact</a></li>
    </ul>
</nav>
<main>
    Main content goes here...
</main>
```

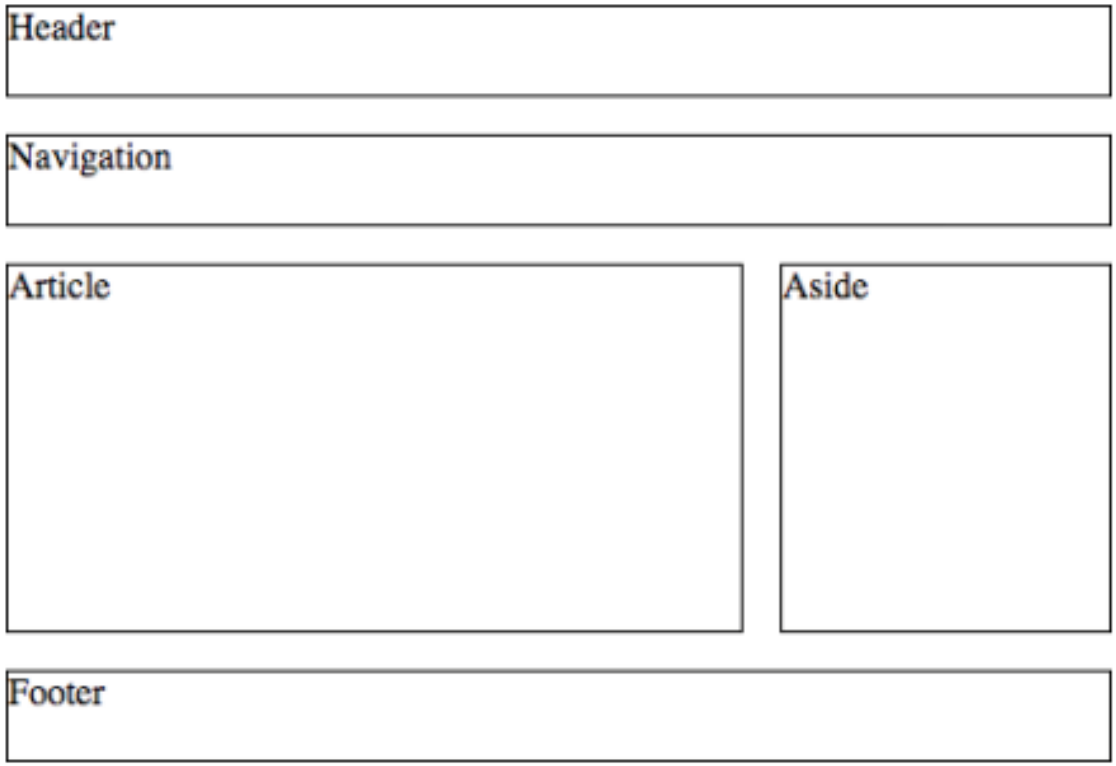
Приклад 2

```
body {
    margin: 2rem;
    width: 30rem;
}
header {
    height: 2.5rem;
    border: 1px solid black;
}
nav {
    height: 2.5rem;
    margin-top: 1rem;
    border: 1px solid black;
}
main {
    margin-top: 1rem;
    height: 10rem;
}
```

```
article {
    float: left;
    margin-right: 1rem;
    width: 20rem;
    height: 100%;
    border: 1px solid black;
}
aside {
    float: right;
    width: 9rem;
    height: 100%;
    border: 1px solid black;
}
footer {
    clear: both;
    height: 2.5rem;
    margin-top: 1rem;
    border: 1px solid black;
}
```



```
<header>
   Header
</header>
<nav>
   Navigation
</nav>
<main>
   <article>
      Article
   </article>
   <aside>
      Aside
   </aside>
</main>
<footer>
   Footer
</footer>
```



2) Laying Out with Inline Blocks

Коли ви перетворюєте елемент на вбудований блок (додаючи **display: inline-block** до правил стилю елемента), відбувається одне з двох речей:

- Якщо ви працюєте з inline елементом, цей елемент стає блоком, але він все ще тече *горизонтально* з рештою навколишнього вкладеного вмісту.
- Якщо ви працюєте з **елементом рівня блоку**, цей елемент видаляється з ***вертикального*** потоку сторінки за замовчуванням і тепер тече *горизонтально* з рештою навколишнього вкладеного вмісту.

Ось загальна процедура, яку слід дотримуватися:

1. Працюйте вниз по сторінці, дозволяючи елементам сторінки викладатися за допомогою потоку сторінок за замовчуванням.
2. Коли ви перейдете до двох або більше елементів, які ви хочете бачити поруч, перетворіть їх у вбудовані блоки.

Приклад 1

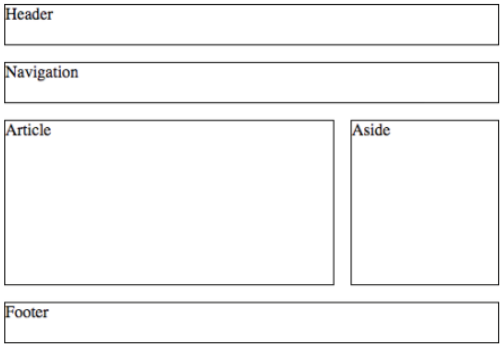
```
nav {
    height: 2.5rem;
    padding-top: .6rem;
    background-color: #ccc;
}
nav ul {
    list-style-type: none;
    padding-left: 1.75rem;
}
nav li {
    display: inline-block;
    padding-right: 1.75rem;
}
main {
    margin-top: 1rem;
}
```

```
<nav>
    <ul>
        <li><a
    href="#">Home</a></li>
        <li><a
    href="#">Blog</a></li>
        <li><a
    href="#">Store</a></li>
        <li><a
    href="#">About</a></li>
        <li><a
    href="#">Contact</a></li>
    </ul>
</nav>
<main>
    Main content goes here...
</main>
```

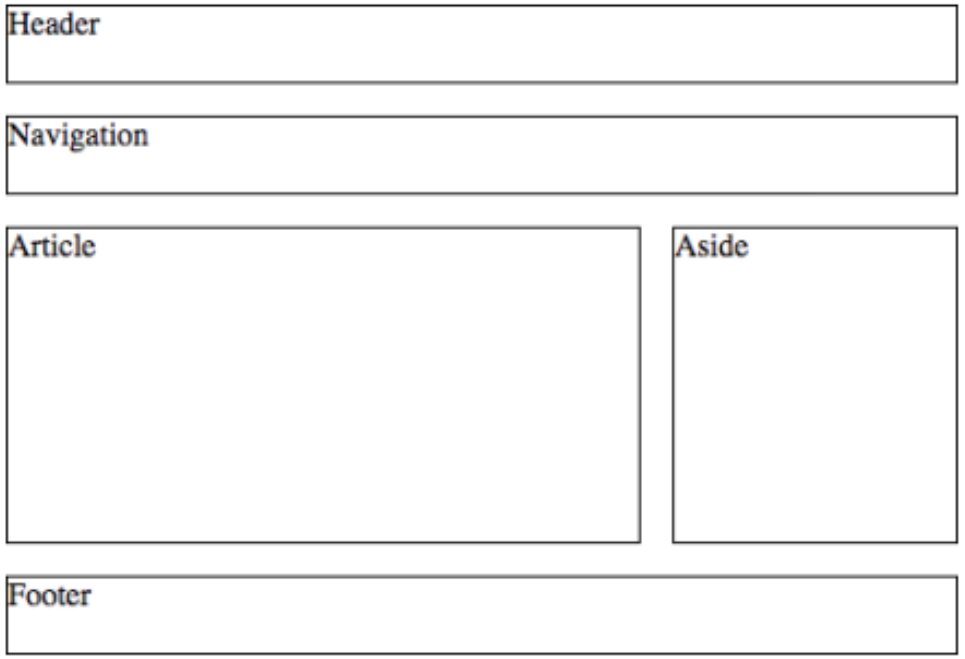
Приклад 2

```
body {
    margin: 2rem;
    width: 30rem;
}
header {
    height: 2.5rem;
    border: 1px solid black;
}
nav {
    height: 2.5rem;
    margin-top: 1rem;
    border: 1px solid black;
}
main {
    margin-top: 1rem;
    height: 10rem;
}
```

```
article {
    display: inline-block;
    margin-right: 1rem;
    width: 20rem;
    height: 100%;
    border: 1px solid black;
}
aside {
    display: inline-block;
    width: 9rem;
    height: 100%;
    border: 1px solid black;
}
footer {
    height: 2.5rem;
    margin-top: 1rem;
    border: 1px solid black;
}
```



```
<header>
    Header
</header>
<nav>
    Navigation
</nav>
<main>
    <article>
        Article
    </article><aside>
        Aside
    </aside>
</main>
<footer>
    Footer
</footer>
```



3) Створення гнучких макетів за допомогою Flexbox

- Коли ви використовуєте або floats , або inline blocks для макета сторінки, є кілька *бананових шкірок* на шляху, які можуть вас зіштовхнути, включаючи забуття очистити floats та забувши переконатися, що між двома вбудованими блоками немає пробілу.
- Однак поза цими ***простими неприємностями*** також є кілька речей, з якими макети на плавучих або вбудованих блоках мають **проблеми**:

1. Дуже важко отримати вміст елемента, **розташованим по центру вертикально** в контейнері елемента.
2. Дуже важко отримати елементи, **рівномірно розташовані горизонтально** по всій ширині (або вертикально по всій висоті) батьківського контейнера.
3. Дуже важко знайти елемент **нижнього колонтитула**, який з'явиться **внизу** вмістової області веб-переглядача.

- На щастя, ці неприємності зникають, якщо використовувати технологію CSS під назвою W3C Flexible Box Layout Module або скорочено flexbox.
- Ключовим тут є "гнучка" частина імені.
- На відміну від потоку сторінок за замовчуванням та макетів, які використовують floats та inline blocks, усі вони рендерують вміст за допомогою жорстких блоків,
- flexbox надає вміст за допомогою контейнерів, які можуть зростати (grow) та зменшуватися (shrink) - тут мова йде і про ширину та висоту - у відповідь на зміну вмісту або розміру вікна веб-переглядача.
- Але flexbox також пропонує потужні властивості, які дозволяють легко розміщувати (lay out), вирівнювати (align), розподіляти та розміщувати дочірні елементи батьківського контейнера.

Flex контейнер і Flex items

Перше, що вам потрібно знати, це те, що **flexbox** ділить світ на дві категорії:

» **Flex container:** Це елемент рівня блоку, який виступає в ролі батьківського для гнучких елементів всередині нього.

» **Flex items:** Це елементи, що знаходяться в гнучкому контейнері.

Setting up the flex container

Щоб позначити елемент як контейнер **flex**, вам потрібно встановите його властивість display у **flex** (або inline-flex):

```
container {
    display: flex;
}
```

Дочірні елементи автоматично стають flex items

- Flexbox - це одновимірний інструмент компонування, що означає, що гнучкі елементи розташовуються всередині їх гнучких контейнерів або **по горизонталі** - тобто по рядку - **або по вертикалі** - тобто в стовпчику.
- Цей напрямок називається первинною віссю (primary axis), і ви задаєте його за допомогою властивості flex-direction:

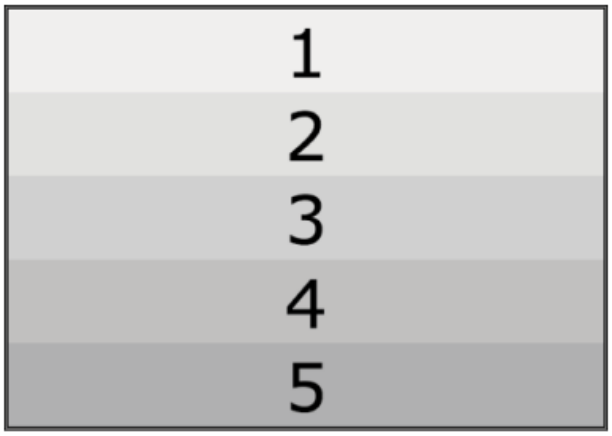
```
element {
    display: flex;
    flex-direction: row | row-reverse | column | column-reverse;
}
```

Вісь, яка перпендикулярна первинній осі, називається вторинною віссю (secondary axis).

Приклад 1

```
.container {
    border: 5px double black;
}
.item {
    border: 1px solid black;
    padding: .1rem;
    font-family: "Verdana", sans-
    serif;
    font-size: 5rem;
    text-align: center;
}
.item1 {
    background-color: rgb(240,
    240, 240);
}
.item2 {
    background-color: rgb(224,
    224, 224);
}
.item3 {
    background-color: rgb(208,
    208, 208);
}
.item4 {
    background-color: rgb(192,
    192, 192);
}
.item5 {
    background-color: rgb(176,
    176, 176);
}
```

```
<div class="container">
    <div class="item item1">1</div>
    <div class="item item2">2</div>
    <div class="item item3">3</div>
    <div class="item item4">4</div>
    <div class="item item5">5</div>
</div>
```



Тепер конфігуруйте батьківський **div** як гнучкий контейнер з горизонтальною первинною віссю:

```
.container {
    display: flex;
    flex-direction: row;
    border: 5px double black;
}
```

Якщо замінити flex на inline-flex :

Вирівнювання flex items уздовж первинної осі

- За замовчуванням гнучкі елементи розташовуються разом з лівої сторони контейнера.
- Це можна змінити, змінивши значення властивості justify-content:

```
container {
    display: flex;
    justify-content: flex-start | flex-end | center | space-between | space-around;
}
```

У таблиці показано кожне з можливих значень властивості justify-content, коли первинна вісь горизонтальна.

Aligning Flex Items along the Primary Axis

justify-content	Example
flex-start	
flex-end	
center	
space-between	
space-around	

Вирівнювання flex items вздовж вторинної осі

- За замовчуванням елементи **flex** завжди займають всю висоту контейнера **flex**, але ви можете отримати інше вирівнювання вторинної осі, змінивши значення властивості **align-items**:

```
container {
    display: flex;
    align-items: stretch | flex-start | flex-end | center | baseline;
}
```

У таблиці показано кожне з можливих значень властивості align-items, коли вторинна вісь вертикальна.

Aligning Flex Items along the Secondary Axis

align-items	Example
stretch	
flex-start	
flex-end	
center	
baseline	

Laying out a navigation bar with flexbox

```
nav {
    background-color: #ccc;
}
nav ul {
    display: flex;
    justify-content: space-around;
    align-items: center;
    height: 2.5rem;
    list-style-type: none;
}
main {
    margin-top: 1rem;
}
```

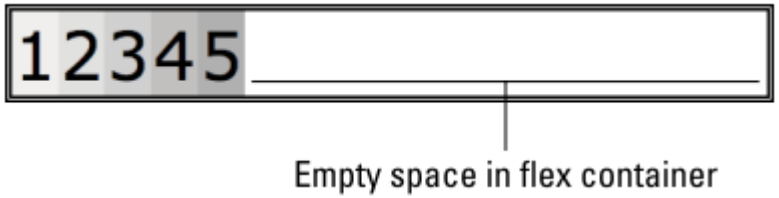
```
<nav>
    <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Blog</a></li>
        <li><a href="#">Store</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
</nav>
<main>
    Main content goes here...
</main>
```

Збільшення flex items

- За замовчуванням, коли ви встановлюєте властивість justify-content на flex-start, flex-end або center, елементи flex займають лише стільки місця вздовж первинної осі, скільки потрібно для їх вмісту.
- Це чудово, але це часто залишає купу **порожнього простору** в гнучкому контейнері.
- Цікаво, що одне із значень “flex” у flexbox полягає в тому, що ви можете змусити один або декілька елементів **зростати**, щоб заповнити цей порожній простір.
- Щоб налаштувати елемент **flex** для зростання, встановлюють властивість flex-grow для елемента:

```
item {
    flex-grow: value;
}
```

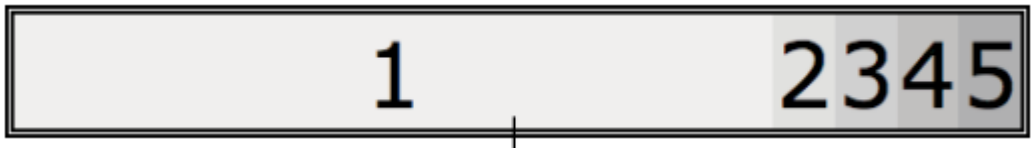
- Тут value - це число, що більше або дорівнює 0. Значення за замовчуванням дорівнює 0, що повідомляє браузеру не збільшувати елементи flex. Це зазвичай призводить до порожнього простору в гнучкому контейнері, як показано на малюнку.



Для позитивних значень flex-grow слід враховувати три сценарії:

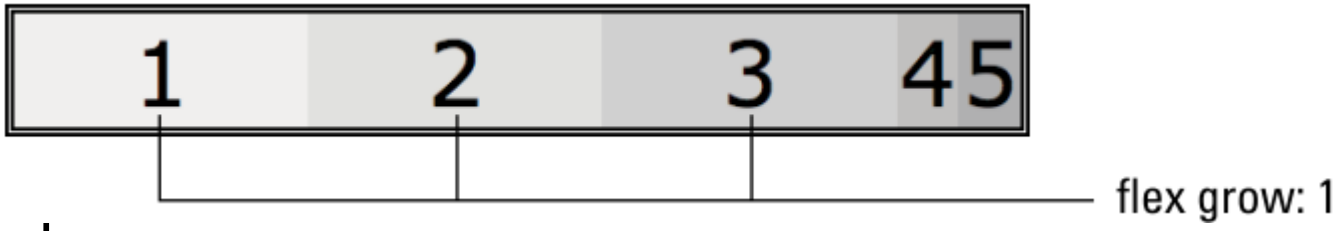
» **Ви присвоюєте позитивне значення flex-grow лише одному елементу**.

```
.item1 {
    flex-grow: 1;
}
```



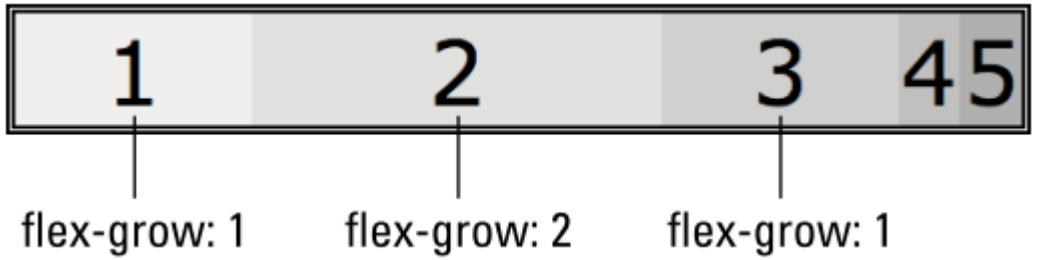
» **Ви присвоюєте одне і те ж позитивне значення flex-grow для двох або більше гнучких елементів.**

```
.item1,
.item2,
.item3 {
    flex-grow: 1;
}
```



» **Ви присвоюєте різну позитивну величину flex-grow для двох або більше гнучких елементів.**

```
.item1 {
    flex-grow: 1;
}
.item2 {
    flex-grow: 2;
}
.item3 {
    flex-grow: 1;
}
```



Allowing flex items to shrink

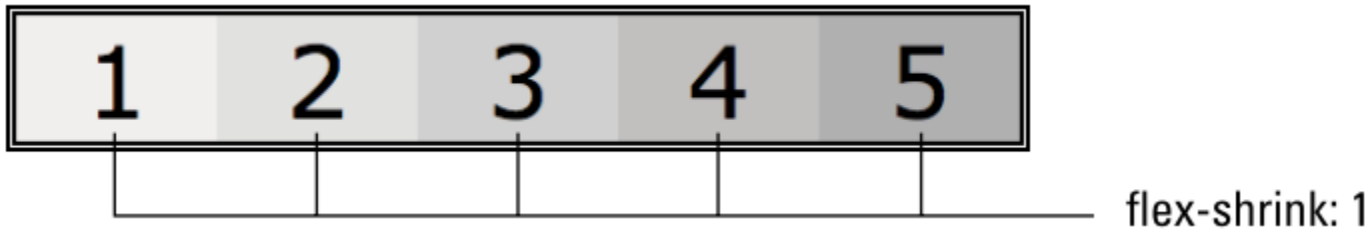
- Гнучкість flexbox означає не тільки те, що гнучкі елементи можуть зростати, заповнюючи порожній простір гнучких контейнерів, але і вони можуть **скорочуватися**, якщо в контейнері flex не вистачає місця для розміщення елементів.
- Стискання гнучких елементів, що вміщуються всередині їх контейнера, є типовою поведінкою flexbox, але ви отримуєте міру контролю над тим, ***які елементи*** зменшуються та на скільки, використовуючи властивість flex-shrink для елемента flex.

```
item {
    flex-shrink: value;
}
```

- Тут value - це число, що більше або дорівнює 0. Значення за замовчуванням дорівнює 1, що дає змогу браузеру зменшити всі гнучкі елементи в рівній мірі, щоб вони вмістилися всередині контейнера flex.

```
.container {
    display: flex;
    width: 500px;
    border: 5px double black;
}
.item {
    width: 200px;
}
```

```
<div class="container">
    <div class="item item1">1</div>
    <div class="item item2">2</div>
    <div class="item item3">3</div>
    <div class="item item4">4</div>
    <div class="item item5">5</div>
</div>
```



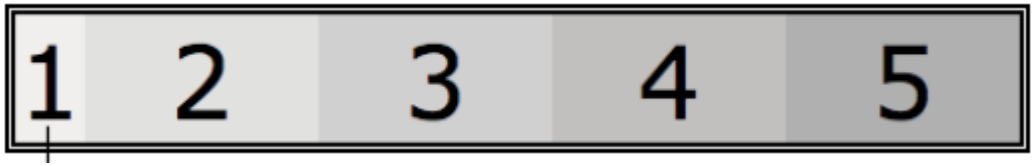
- Для позитивних значень **flex-shrink**, у вас є три способи контролювати усадку елемента flex:

» Призначити елементу значення flex-shrink між 0 і 1.

```
.item1 {
    flex-shrink: .5;
}
```

» Призначити елементу значення flex-shrink більше1.

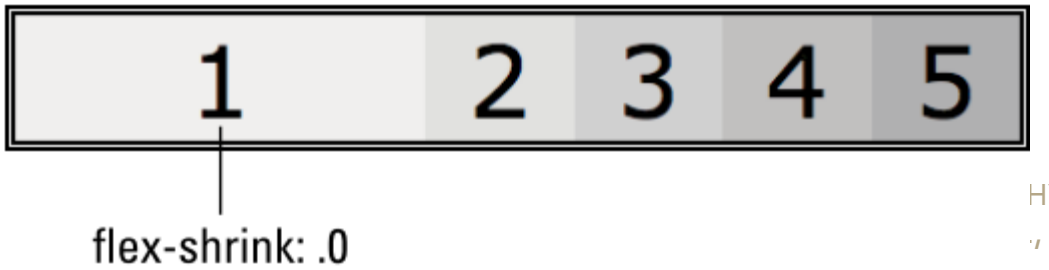
```
.item1 {
    flex-shrink: 2;
}
```



» Призначити елементу значення flex-shrink = 0.

Веб-переглядач не зменшує елемент.

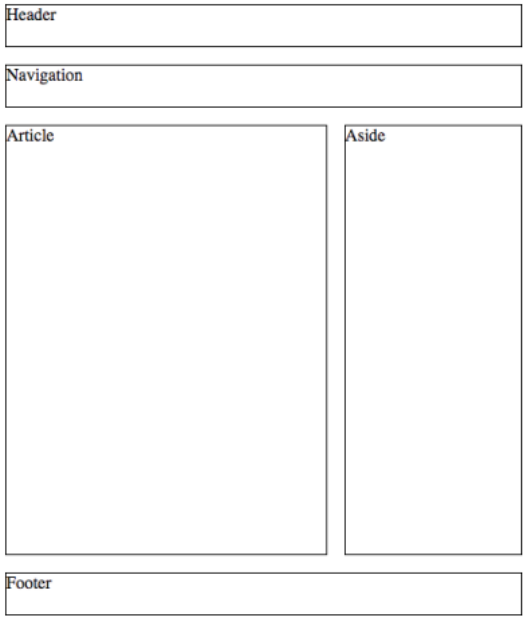
```
.item1 {
    flex-shrink: 0;
}
```



Laying out content columns with flexbox

- Flexbox найкраще працює, коли ви використовуєте його для розміщення компонентів по одному виміру, але це не означає, що ви не можете використовувати його для макетування всієї сторінки.
- Поки структура сторінки відносно проста, флексбокс відмінно підходить для розміщення елементів **як горизонтально, так і вертикально**.

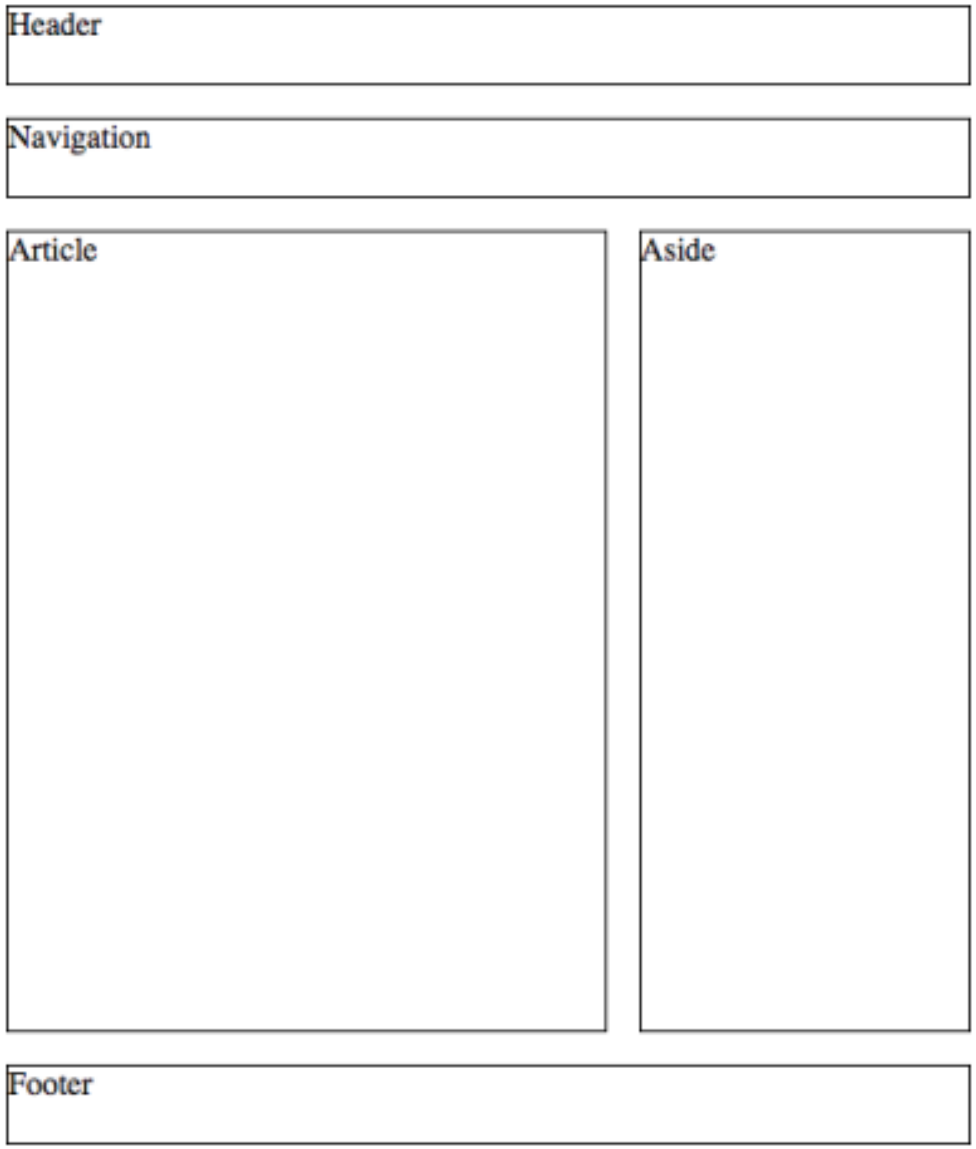
```
body {
    display: flex;
    flex-direction: column;
    width: 30rem;
    min-height: 100vh;
}
```



```
header {
    height: 2.5rem;
    border: 1px solid black;
}
nav {
    height: 2.5rem;
    margin-top: 1rem;
    border: 1px solid black;
}
main {
    flex-grow: 1;
    display: flex;
    margin-top: 1rem;
}
```

```
article {
    flex-grow: 1;
    margin-right: 1rem;
    border: 1px solid black;
    overflow-y: auto;
}
aside {
    flex-grow: 0;
    flex-shrink: 0;
    flex-basis: 10rem;
    border: 1px solid black;
}
footer {
    height: 2.5rem;
    margin-top: 1rem;
    border: 1px solid black;
}
```

```
<body>
    <header>
        Header
    </header>
    <nav>
        Navigation
    </nav>
    <main>
        <article>
            Article
        </article>
        <aside>
            Aside
        </aside>
    </main>
    <footer>
        Footer
    </footer>
</body>
```



Пояснення:

- Тег **<body>** налаштований як контейнер flex, і цей контейнер стилізований у flex-direction: column, щоб створити вертикальну первинну вісь для сторінки в цілому.
- Елемент **<body>** має властивість min-height, встановлену на 100vh, завдяки чому flex container завжди займає принаймні всю висоту області вмісту браузера.
- Всім елементам header, nav, та footer надаються явні значення висоти.
- Елемент main стилізований як flex-grow: 1, що повідомляє браузеру збільшувати main елемент вертикально, поки він не займе порожній простір у контейнері flex. Це також гарантує, що елемент footer з'являється в нижній частині області вмісту, навіть якщо не вистачає вмісту для заповнення основного елемента.
- main елемент є також flex container, стилізований як flex-direction: row для створення горизонтальної первинної осі.

- Всередині main flex–контейнера елементу article надається flex-grow: 1, тож він зростає в міру необхідності, щоб зайняти ширину основного елемента, що залишилася (тобто після того, як буде врахована ширина елемента aside).
- Щоб отримати бічну панель фіксованої ширини, правило елемента aside має як flex-grow, так і flex-shrink, встановлені в 0, а також включає в себе декларацію flex-basis: 10rem. Властивість flex-basis надає браузеру запропоноване *початкове значення* для розміру елемента. У цьому випадку, як при flex-grow, так і при flex-shrink, встановлених в 0, значення flex-basis діє як фіксована ширина.

На замітку (порада):

- Існує скорочена властивість, яка називається flex, яку ви можете використовувати для об'єднання flex-grow, flex-shrink, та flex-base в одне оголошення:

```
item {
  flex: grow-value shrink-value basis-value;
}
```

Наприклад, ми можемо переписати правило для aside елемента у наведеному вище прикладі так:

```
aside {
    flex: 0 0 10rem;
    border: 1px solid black;
}
```

Підтримка Flexbox браузерами

- Хороша новина полягає в тому, що всі основні веб-браузери, як настільні, так і мобільні, підтримують flexbox.
- Погана новина полягає в тому, що вони не завжди підтримували flexbox або, якщо бути точним, вони підтримували його, але лише з тим, що відоме як ***префікси постачальника***.
- Префікс постачальника - це мітка, характерна для кожного браузера - наприклад, -webkit- для браузерів, які використовують механізм візуалізації сторінки WebKit (включаючи Chrome та Safari), -moz- для Firefox та -ms- для Microsoft Edge та Internet Explorer - що дозволяло браузеру реалізувати функцію CSS.

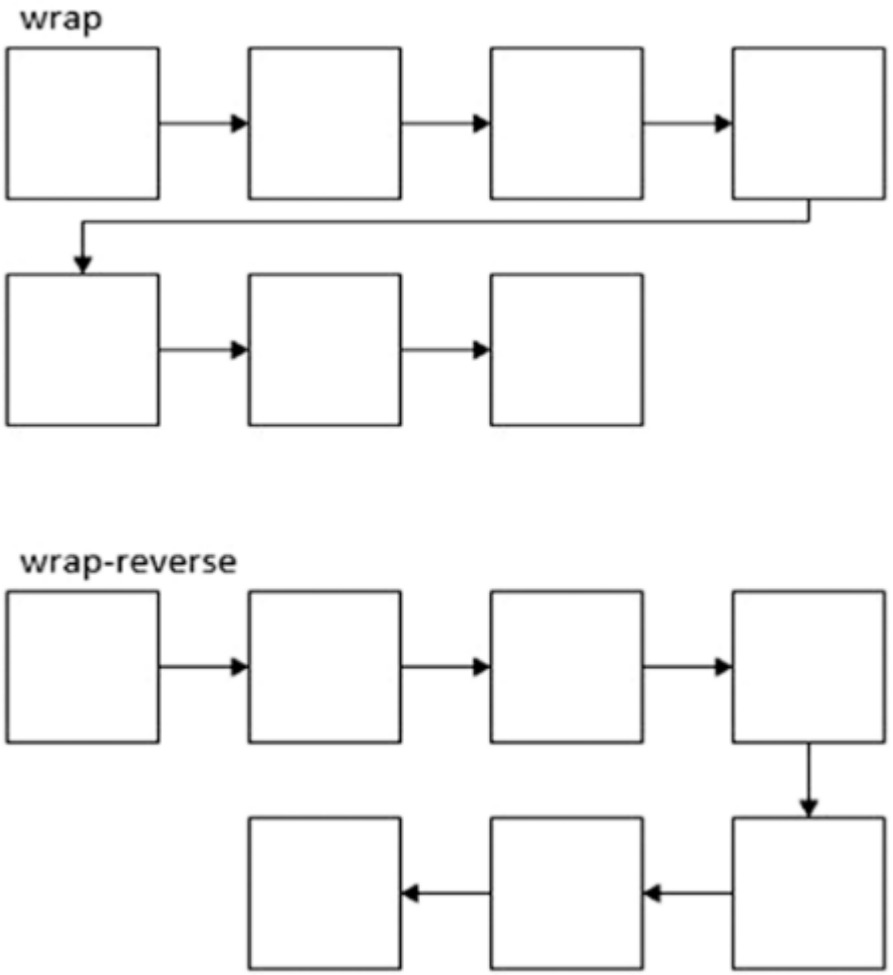
- Отже, хоча декларація, наприклад display: flex, буде працювати чудово у приблизно 90 відсотках сьогоднішніх браузерів, щоб охопити решту, вам потрібно включити попередньо встановлені версії тієї ж декларації:

```
container {
    display: -webkit-box;
    display: -ms-flexbox;
    display: flex;
}
```

Flex Wrap

- Коли flex-direction є row (або row-reverse), елементи в контейнері течуть по горизонталі так, ніби використовується display:inline; і коли ширина контейнера заповнена, наступні елементи перегортають в наступний ряд.
- Атрибут **flex-wrap** керує тим, як і як це робиться. Можливі три значення:

1. nowrap - (default) Елементи відображаються в одному рядку (або стовпці).
2. wrap - Елементи будуть переведені на наступний рядок або стовпець, використовуючи той самий напрямок, що і початковий.
3. wrap-reverse - Елементи переносять на наступний рядок або стовпець, але роблять це у зворотному порядку.



- Demonstrating the wrapping options

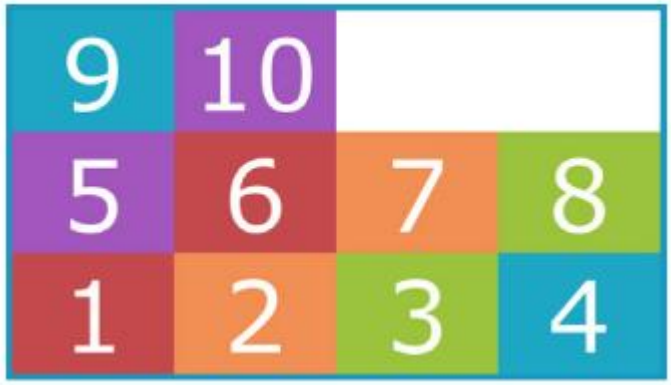
Приклад

```
<div id="container">
   <div class="box box1">1</div>
   <!-- more boxes here -->
   <div class="box box10">10</div>
</div>

#container {
   display: flex;
   flex-direction: row;
   flex-wrap: wrap;
}
.box {
    width: 25%;
}
```

flex-wrap: wrap;

flex-wrap: wrap-reverse;



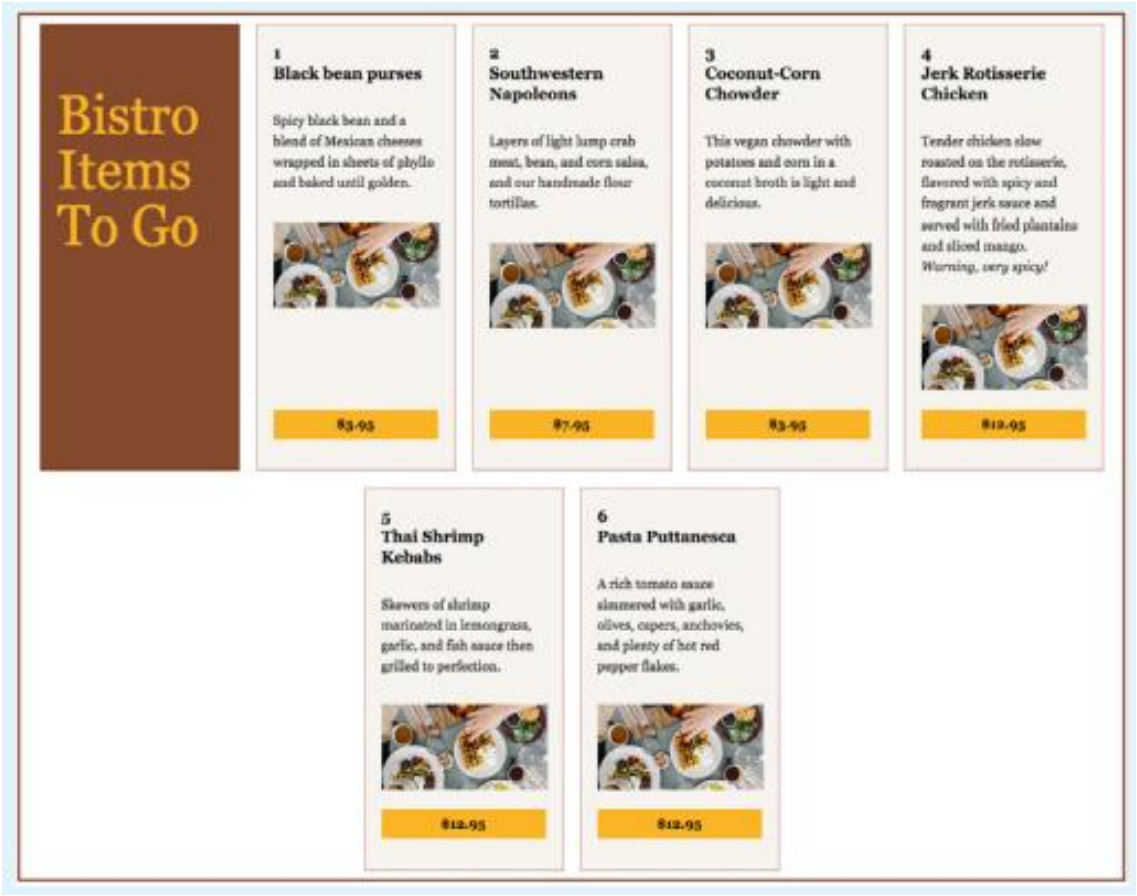
EXERCISE 16-1. стор.427
3 книги Learning Web Design 5th 2018

```
nav ul {
margin: 0;
  padding: 0;
  list-style-type: none;
  display: flex;
  justify-content: center;
}

nav ul li a {
  display: block;
  border: 1px solid;
  border-radius: .5em;
  padding: .5em 1em;
  margin: .5em;
}
```

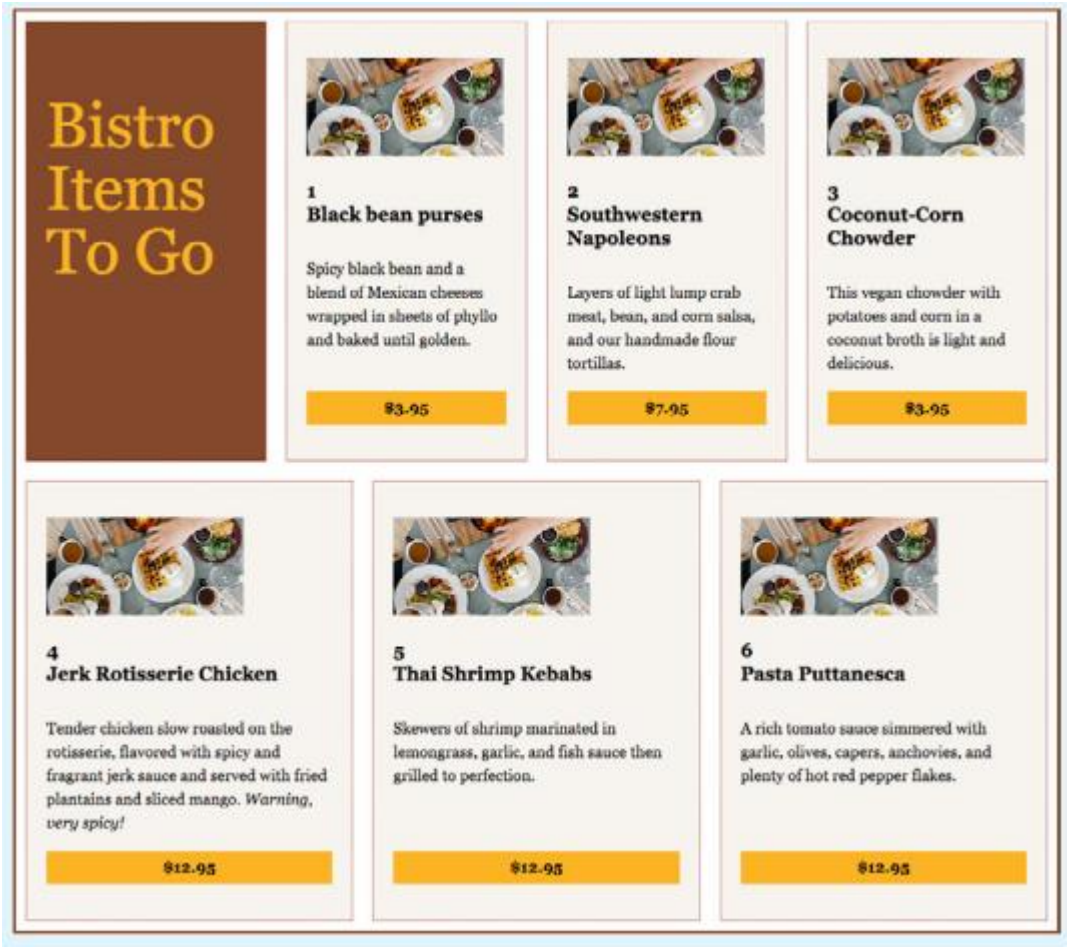
FIGURE 16-7. The list of links is now styled as a horizontal menu bar.

EXERCISE 16-2. p.434
3 книги Learning Web Design 5th 2018



EXERCISE 16-2. p.445

3 книги Learning Web Design 5th 2018



4) Формування загального макета сторінки за допомогою CSS Grid

- Однією з найбільш захоплюючих і очікуваних розробок у недавній історії CSS є поява технології під назвою CSS Grid.
- Специфікація Grid дає вам простий спосіб розділити контейнер на один або кілька рядків і один або кілька стовпців - тобто як **сітку** -, а потім необов'язково призначити елементи контейнера певним ділянкам сітки.
- За допомогою CSS Grid ви можете давати веб-браузерам такі інструкції, як:

» Встановити тег **<body>** як сітку з чотирма рядками та трьома стовпцями.

» Помістити елемент header в перший рядок і зробити його прольотом (span) у всіх трьох стовпцях.

» Помістити елемент **nav** в другий рядок і зробити його span на всіх трьох стовпцях.

» Помістити елемент article в третій рядок, стовпці один і два.

» Помістіть елемент aside у третьому рядку, стовпець три.

» Помістити елемент footer в четвертий рядок і зробити його прольотом (span) у всіх трьох стовпцях.

Перш ніж навчитися робити все це та багато іншого, потрібно знати, що Grid використовує дві категорії елементів:

- Grid container: Це елемент рівня блоку, який виступає батьківським для елементів всередині нього і налаштовується за допомогою заданої кількості рядків і стовпців.
- Grid items: Це елементи, що знаходяться в контейнері сітки і які ви призначаєте (або браузер призначає автоматично) певним частинам сітки.

Налаштування grid container

- Щоб позначити елемент як grid container, ви встановлюєте його властивість display в grid :

```
container {
    display: grid;
}
```

Коли перший крок завершений, діти елемента автоматично стають елементами сітки (grid items).

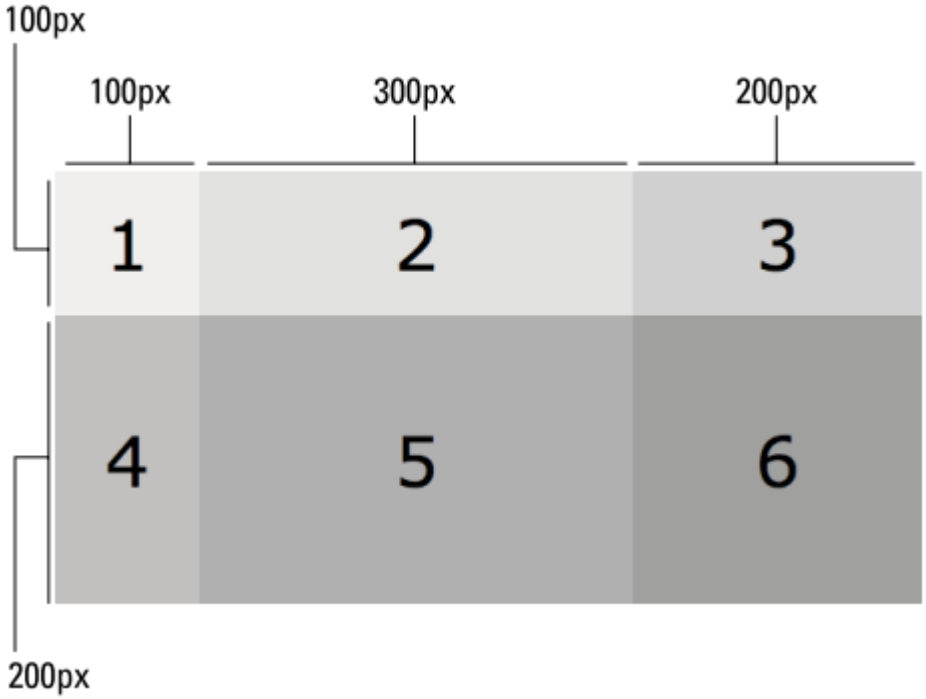
Вказання рядків та стовпців сітки

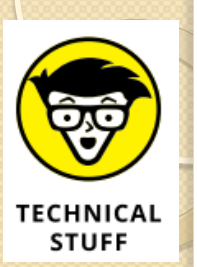
```
container {
    display: grid;
    grid-template-columns: column-values;
    grid-template-rows: row-values
}
```

Приклад 1 (part1.html)

```
.container {
    display: grid;
    grid-template-columns: 100px 300px 200px;
    grid-template-rows: 100px 200px;
}
```

```
<div class="container">
    <div class="item item1">1</div>
    <div class="item item2">2</div>
    <div class="item item3">3</div>
    <div class="item item4">4</div>
    <div class="item item5">5</div>
    <div class="item item6">6</div>
</div>
```





- Ви також можете вказати розмір стовпця або рядка, використовуючи новий unit під назвою fr, який характерний для Grid і являє собою частину вільного простору, доступного в контейнері сітки, або горизонтально (для стовпців), або вертикально (для рядків).
- Наприклад, якщо призначити одному стовпцю 1fr простору та іншому стовпцю 2fr, браузер надає третині вільного горизонтального простору першому стовпцю і дві третини горизонтального вільного простору другому стовпцю.

- Якщо ви не встановите властивість grid-template-rows , браузер автоматично налаштовує висоту рядків на основі висоти найвищого елемента в кожному рядку.

Створення зазорів сітки (grid gaps)

- За замовчуванням браузер не містить горизонтального проміжку між кожним стовпцем або вертикального пробілу між кожним рядком. Якщо ви встановити зазори між елементами сітки, ви можете додати властивості grid-column-gap та grid-row-gap до контейнера сітки:

```
container {
    display: grid;
    grid-column-gap: column-gap-value;
    grid-row-gap: row-gap-value
}
```

Наприклад:

```
.container {
    display: grid;
    grid-template-columns: 100px 300px 200px;
    grid-template-rows: 100px 200px;
    grid-column-gap: 10px;
    grid-row-gap: 15px;
}
```

- Існує скорочена властивість під назвою grid-gap, яку ви можете використовувати для об'єднання grid-column-gap та grid-row-gap в одне оголошення:

```
container {
    display: grid;
    grid-gap: column-gap-value row-gap-value;
}
```

- Нещодавно CSS Grid overlords заявив, що назви властивостей, пов’язаних з gap, в майбутньому будуть змінюватися:

Current Name	Future Name
grid-column-gap	column-gap
grid-row-gap	row-gap
grid-gap	gap

Призначення елементів сітки рядкам і стовпцям

- Замість того, щоб веб-браузер автоматично заповнював сітку, ви можете взяти під контроль процес і призначити елементи вашої сітки певним рядкам і стовпцям. Для кожного елемента сітки ви вказуєте чотири значення:

```
item {
    grid-column-start: column-start-value;
    grid-column-end: column-end-value;
    grid-row-start: row-start-value;
    grid-row-end: row-end-value;
}
```

- grid-column-start: Число, яке вказує стовпчик, з якого починається елемент.
- grid-column-end: Число, яке вказує стовпчик, перед яким елемент закінчується. Наприклад, якщо для grid-column-end встановлено 4, елемент сітки закінчується в стовпці 3. Деякі примітки:

- Якщо цю властивість опустити, елемент використовує лише початковий стовпець.
- Якщо ви використовуєте ключове слово end, то елемент працює від його початкового стовпця до останнього стовпця в сітці.
- Ви можете скористатися ключовим словом span, за яким слідує пробіл, а потім число, яке визначає кількість стовпців, для яких елемент повинен проходити через сітку. Наприклад, наступні два набори декларацій еквівалентні:

```
grid-column-start: 1;
grid-column-end: 4;
grid-column-start: 1;
grid-column-end: span 3;
```

- grid-row-start: Число, яке вказує рядок, з якого починається елемент

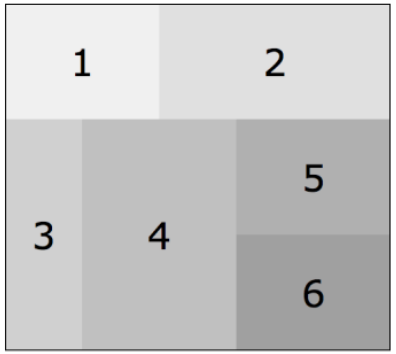
- grid-row-end: Число, яке вказує рядок, перед яким елемент закінчується. Наприклад, якщо для grid-row-end встановлено 3, елемент сітки закінчується у рядку 2. Деякі примітки:
  - Якщо цю властивість опустити, елемент використовує лише початковий рядок.
  - Якщо ви використовуєте ключове слово **end**, то елемент працює від його початкового рядка до останнього рядка в сітці.
  - Ви можете скористатися ключовим словом **span**, за яким слід пробіл, а потім число, яке вказує кількість рядків, на які подовжено елемент. Наприклад, наступні два набори декларацій еквівалентні:

```
grid-row-start: 2;
grid-row-end: 4;
grid-row-start: 2;
grid-row-end: span 2;
```

Приклад

```
.container {
    display: grid;
    grid-template-columns:
    repeat(5, 100px);
    grid-template-rows: repeat(3,
    150px);
}
.item1 {
    grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 1;
    grid-row-end: 1;
}
.item2 {
    grid-column-start: 3;
    grid-column-end: span 3;
    grid-row-start: 1;
    grid-row-end: 1;
}
```

```
.item3 {
    grid-column-start: 1;
    grid-column-end: 1;
    grid-row-start: 2;
    grid-row-end: end;
}
.item4 {
    grid-column-start: 2;
    grid-column-end: 4;
    grid-row-start: 2;
    grid-row-end: end;
}
```



```
.item5 {
    grid-column-start: 4;
    grid-column-end: span 2;
    grid-row-start: 2;
    grid-row-end: 2;
}
.item6 {
    grid-column-start: 4;
    grid-column-end: span 2;
    grid-row-start: 3;
    grid-row-end: 3;
}
```


```
<div class="container">
    <div class="item item1">1</div>
    <div class="item item2">2</div>
    <div class="item item3">3</div>
    <div class="item item4">4</div>
    <div class="item item5">5</div>
    <div class="item item6">6</div>
</div>
```

Вирівнювання елементів сітки (grid items)

- CSS Grid пропонує кілька властивостей, які можна використовувати для вирівнювання елементів сітки. Для grid container у вас є justify-items та align-items властивості:

```
container {
    justify-items: start | end | center | stretch;
    align-items: start | end | center | stretch;
}
```

- justify-items: Вирівнює вміст всередині кожного елемента сітки горизонтально. Ви можете вирівняти елементи ліворуч (start), праворуч (end), посередині (center) або поперек ширини елемента (stretch; це значення за замовчуванням).
- align-items: Вирівнює вміст всередині кожного елемента сітки вертикально. Ви можете вирівняти елементи вгорі (start), знизу (end), посередині (center) або по всій висоті елемента (stretch; це значення за замовчуванням).

- Для елемента сітки ви маєте властивості justify-self та align-self:

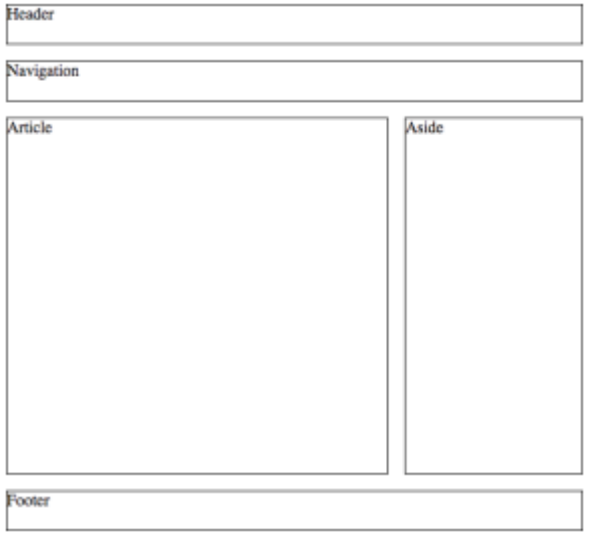
```
item {
    justify-self: start | end | center | stretch;
    align-self: start | end | center | stretch;
}
```

- justify-self: Вирівняє вміст всередині елемента сітки горизонтально. Ви можете вирівняти елемент ліворуч (start), праворуч (end), посередині (center) або поперек ширини елемента (stretch; це значення за замовчуванням).
- align-self: Вирівнює вміст всередині елемента сітки вертикально. Ви можете вирівняти елемент вгорі (start), знизу (end), посередині (center) або по всій висоті елемента (stretch; це значення за замовчуванням).

Laying out content columns with Grid

- Як двовимірна система компонування, Grid ідеально підходить для викладання всієї сторінки.
- Розглянемо класичний макет сторінки: заголовок та панель навігації вгорі сторінки, стаття з бічною панеллю поруч та нижній колонтитул внизу сторінки.
- Ось код Grid, що створює макет, показаний на малюнку:

```
body {
    display: grid;
    grid-template-columns: 1fr 10rem;
    grid-template-rows: 2.5rem 2.5rem 1fr 2.5rem;
    grid-gap: 1rem 1rem;
    min-height: 100vh;
}
```



```
header {
    grid-column: 1 / end;
    grid-row: 1;
    border: 1px solid black;
}
nav {
    grid-column: 1 / end;
    grid-row: 2;
    border: 1px solid black;
}
article {
    grid-column: 1;
    grid-row: 3;
    border: 1px solid black;
}
aside {
    grid-column: 2 / end;
    grid-row: 3;
    border: 1px solid black;
}
```

```
footer {
    grid-column: 1 / end;
    grid-row: 4;
    border: 1px solid black;
}
```

```
<body>
    <header>
        Header
    </header>
    <nav>
        Navigation
    </nav>
    <article>
        Article
    </article>
    <aside>
        Aside
    </aside>
    <footer>
        Footer
    </footer>
</body>
```

Пояснення:

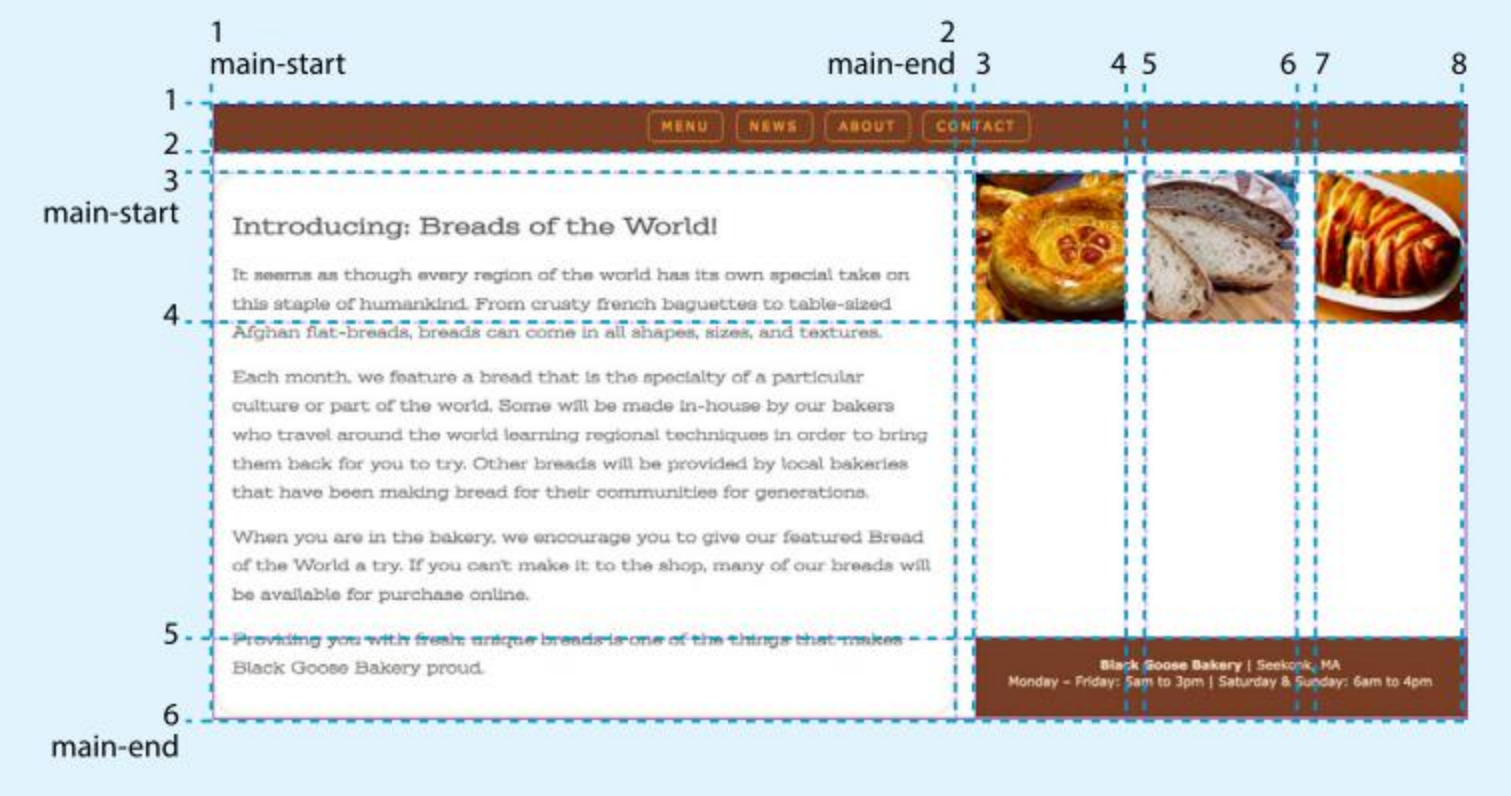
- Тег <body> встановлюється як grid container, і цей контейнер стилізований з двох стовпців і чотирьох рядків.
- Елемент body має властивість min-height, встановлену в 100vh, завдяки чому контейнер з сіткою завжди займає принаймні всю висоту області вмісту браузера.
- Усі елементи header, nav та footer простягаються від першого стовпця до кінця сітки та їм присвоюються рядки 1, 2 та 4 відповідно.
- Ця версія класичного макета не включає main елемент, оскільки **CSS Grid** не пропонує механізм вкладання сіток.
- Елемент article використовує лише стовпчик 1 і рядок 3, обидва вони були визначені розміром 1fr, що дозволяє елементу статті займати вільний простір у сітці.
- Елемент aside використовує стовпчик 2, якому було призначено ширину 10rem, тому його ширина фіксована.

Grid browser support

CSS Grid пропонує дві частини дуже хороших новин, коли мова йде про підтримку браузера:

- Усі основні веб-браузери, як на робочому столі, так і на мобільних пристроях, підтримують CSS Grid.
- У вашому CSS-коді не потрібні префікси.
- Nearly 80 percent of browsers support Grid. This is too small a number to build a Grid-only layout.
- About 85 percent of browsers fully support flexbox, although vendor prefixes are required. This is great support, but if you do a flexbox-only layout, about one in seven visitors will see your page in an ugly light.
- All browsers support both the float property and display: inline-block.

EXERCISE 16-4. p.461
3 книги Learning Web Design 5th 2018



EXERCISE 16-6. p.476

3 книги Learning Web Design 5th 2018