RT.Academy

CSS

Cascading Style Sheets

Зміст

Зміст

Початок роботи з CSS

CSS (Cascading Style Sheets, каскадні таблиці стилів) використовується для стилізації і компонування веб-сторінок - наприклад, для зміни шрифту, кольору, розміру та інтервалу вмісту, поділу його на декілька стовпців або додавання анімації та інших декоративних елементів.

На відміну від HTML, який служить для визначення структури і семантики вмісту, CSS відповідає за його зовнішній вигляд і відображення.

CSS - це мова на основі правил: ви задаєте правила, що визначають групи стилів, які повинні застосовуватися до певних елементів або груп елементів на Вашій веб-сторінці.

Детальніше:
https://uk.wikipedia.org/wiki/CSS

Селектори: Селектор за типом

```
h1 {
    color: red;
    font-size: 20px;
}

p, li {
    color: green;
}

li {
    background-color: black;
    color: blue;
}
```

Правило відкривається за допомогою селектора (Selector). Цей селектор вибирає HTML-елемент, який ми збираємося стилізувати. В цьому випадку ми використовуємо заголовки першого рівня (<h1>).

Потім у нас є набір фігурних дужок {}, всередині яких йде перелік властивостей та їх значень через двокрапку (:), що відокремлюються між собою крапкою з комою (;)

Селектори можуть обирати декілька HTML-елементів.

Правила можливо доповнювати і перекривати.

Селектор, що обирає елементи за тегом називається селектор за типом (Type selector)

Селектори: Селектор за класом

```
.my-special-class {
    color: blue;
    font-size: 30px;
}
```

```
<p>Цей текст буде відображено звичайним стилем</p>
<p class="my-special-class">Цей текст буде відображено інакше</p>
<p>Цей текст буде відображено звичайним стилем</p>
<h3 class="my-special-class other-class">Цей текст буде відображено також інакше</h3>
```

Переглянути у JSFiddle

Ви можете додати до будь-якого елемента на сторінці в значення атрибута `class` (у прикладі це значення `my-special-class`) назву CSS-класу та визначити для нього стиль відображення (селектор назви класу має починатися з крапки).
Тепер всі елементи з таким значенням атрибута `class` отримають інакше відображення.

Селектор, що обирає елементи, що відповідають вказаному атрибуту class називається селектор за класом (Class selector).

Вимоги до назви класу:

- мінімум 2 символи довжиною
- будь-які символи з набору a-z, A-Z, цифри, нижнє підкреслення (_) і мінус (-)

Селектори: Селектор за ID

```
#name {
    color: blue;
    font: normal 30px/40px Verdana;
}
```

```
<h2>Цей текст відобразиться звичайним стилем</h2>
<h2 id="name">Цей текст відобразиться інакше</h2>
```

Селектор, що обирає елемент, відповідно до його значення атрибуту id називається селектор за ID (ID selector).

В документі повинен бути тільки один елемент з вказаним ID.

Переглянути у JSFiddle

Селектори: Псевдоклас

```
h2 {
    color: black;
    font: normal 30px/40px Verdana;
}
h2:hover {
    color: blue;
}
```

```
<h2>Цей текст змінить колір з чорного на синій при наведенні вказівника миші</h2>
```

Переглянути у JSFiddle

Псевдоклас в CSS - це ключове слово, що додається до селектора, яке визначає його особливий стан.

Псевдокласи дають можливість стилізувати елемент на основі не тільки структури документа, але і базуючись на зовнішніх факторах. Наприклад:

- історії відвідувань браузера (наприклад :visited)
- стан вмісту (наприклад :checked для деяких елементів форми)
- позиції курсора миші (наприклад :hover, що визначає, чи знаходиться курсор миші над елементом)

Список всіх можливих псевдокласів https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes

Селектори: Всі селектори

`*`	будь-який елемент на сторінці *(універсальний селектор)*	`* { … }`
`#X`	елемент з визначеним ID *(селектор за ID)*	`#container { … }`
`.X`	елементи з вказаним класом *(селектор за класом)*	`.error { … }`
`X`	елементи з вказаним тегом *(селектор за типом)*	`a { … }`

`X:visited`
`X:link`
`X:checked`
	псевдокласи, інші різні умови на елемент	`a:link { … }`
		`a:visited { … }`
		`input[type=radio]:checked { … }`

`X Y`	елементи Y, які є нащадками X *(контекстний селектор)*	`li a { … }`
`.X.Y`	елементи одночасно з двома класами X і Y	`.hello.world { … }`
`X > Y`	тільки безпосередні (прямі) нащадки	`div#container > ul { … }`

`X + Y`	перший правий сусід, Y на тому ж рівні вкладеності, який йде відразу після X *(сусідній селектор, тільки перший елемент)*	`ul + p { … }`
`X ~ Y`	все Y на тому ж рівні вкладеності, які йдуть після X (праві сусіди, все елементи)	`ul ~ p { … }`
`X[href="foo"]`	тільки елементи, у яких є вказаний атрибут href і він дорівнює значенню в лапках *(селектор за атрибутом)*	`a[href="https://google.com"] { … }`
`X:before` `X:after`	псевдокласи, що дозволяють згенерувати контент навколо зазначеного елемента	`.clearfix:before { … }` `.clearfix:after { … }`
`X:first-child` `X:last-child`	псевдоклас для тільки першого/останнього дочірнього елемента	`ul li:first-child { … }` `ul > li:last-child { … }`

Під'єднання CSS: Зовнішня таблиця стилів

Це найпоширеніший і корисний спосіб під’єднання CSS до HTML-документа.
Так ви можете прив'язати CSS відразу до декількох сторінок, що дозволяє стилізувати їх однією таблицею стилів.
У більшості випадків різні сторінки сайту будуть виглядати майже так само, тому ви можете використовувати один і той же набір правил для основного виду.

Зовнішня таблиця стилів - це коли у вас є CSS окремим файлом з розширенням .css і посилання на нього з HTML-елемента <link> в елементі <head>

Можливе під’єднання будь-якої кількості таких файлів, але рекомендується не більше одного.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <h1>Привіт!</h1>
  </body>
</html>
```

Під'єднання CSS: Внутрішня таблиця стилів

У такому випадку CSS поміщений всередині елемента <style>, що міститься всередині HTML <head>

Такий спосіб під’єднання корисний у випадку коли ваші стилі стосуються лише конкретної сторінки і більше не використовуються ніде.

Найбільш поширений варіантом є під’єднання зовнішньої таблиці стилів, стилі якої доповнюють/перекривають внутрішніми стилями поточної сторінки.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS</title>
    <style>
      h1 {
        color: blue;
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <h1>Привіт!</h1>
  </body>
</html>
```

Під'єднання CSS: Вбудовані (inline) стилі

Вбудовані стилі є правилами CSS, які впливають тільки на той елемент, для якого визначено стилі відображення в атрибуті style.

Це не рекомендований варіант і використовувати його необхідно лише у крайніх випадках (наприклад для динамічної зміни кольору і/або позиції елементів на сторінці).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS</title>
  </head>
  <body>
    <h1 style="color: blue;">Привіт!</h1>
  </body>
</html>
```

Завдання #3.1 (1/2)

1. Створіть файл styles.css, що буде містити таблицю стилів (CSS) і підключіть як зовнішню таблицю стилів на кожну HTML-сторінку

2. Додайте правила що праворуч в цю таблицю стилів (styles.css).
   Якщо все виконали коректно, то на всіх сторінках всі елементи <h1> стануть темно-синього кольору та 24px висотою, <h2> - 22px висотою, <h3> - 20px висотою, а <h4>, <h5> та <h6> - 18px висотою.

3. Перевірте п.2 через Chrome DevTools

```css
h1 {
    color: darkblue;
    font-size: 24px;
}

h2 {
    font-size: 22px;
}

h3 {
    font-size: 20px;
}

h4, h5, h6 {
    font-size: 18px;
}
```

Завдання #3.1 (2/2)

4. Додайте на сторінку contacts.html елемент <style> у елемент <head> після елементу <link> з внутрішньої таблицею стилів, в якій змінить колір <h1> на темно-червоний (darkred).

    a. Що відбудеться?
    b. Як це вплине на інші сторінки?
    c. Що відбудеться якщо елемент <style> перенести ПЕРЕД елементом <link>?

5. Додайте на сторінку contacts.html у елемент <h1> вбудований стиль, в якому збільшіть розмір шрифта з 24px до 28px (тільки для цього елементу).

6. Перевірте HTML-сторінки на помилки за допомогою HTML-валідатора validator.w3.org

7. Перевірте файл стилів styles.css на помилки за допомогою CSS-валідатора jigsaw.w3.org/css-validator

Значення властивостей CSS

Кожна властивість, що використовується в CSS, має значення чи набір допустимих значень.

Список типів таких властивостей:

- Число
- Відсотки
- Довжини (абсолютні та відносні)
- Колір

Значення властивостей CSS: Числа

Деякі значення приймають числа без будь-яких одиниць виміру.

Прикладом властивості, що приймає числа без одиниць виміру може служити властивість opacity, яка контролює прозорість елемента. Це властивість приймає числа між 0 (повністю прозоре) і 1 (повністю непрозоре).

```
.box1 {
    opacity: 0.5;
}

.box2 {
    padding: 0;
}
```

Значення властивостей CSS: Відсотки

Відсотки завжди встановлюються щодо деякого іншого значення.

Наприклад, якщо ви встановите font-size елемента у відсотках, то це буде відсоток від font-size батьківського елемента. Якщо ви використовуєте відсоток для значення width (ширина), то це буде відсоток від width батьківського елемента.

```
.fs20 {
    font-size: 20%;
}

.w40 {
    width: 40%;
}
```

Значення властивостей CSS: Довжини

Абсолютні одиниці довжини

Всі одиниці абсолютної довжини не є відносними до чого-небудь і зазвичай вважаються завжди однакового розміру. Більшість з цих значень більше корисні при використанні друку, ніж для виведення на екран. Наприклад, ми зазвичай не використовуємо cm (сантиметри) на екрані. Єдине значення яке ви в основному будете використовувати це px (пікселі).

Одиниця	Назва	Еквівалент
`cm`	Centimeters/сантиметри	`1cm = 96px/2.54`
`mm`	Millimeters/міліметри	`1mm = 1/10th of 1cm`
`Q`	Quarter-millimeters/Чверть-міліметра	`1Q = 1/40th of 1cm`
`in`	Inches/дюйми	`1in = 2.54cm = 96px`
`pc`	Picas/піки	`1pc = 1/16th of 1in`
`pt`	Points/точки	`1pt = 1/72th of 1in`
`px`	Pixels/пікселі	`1px = 1/96th of 1in`

Значення властивостей CSS: Довжини

Одиниці відносної довжини

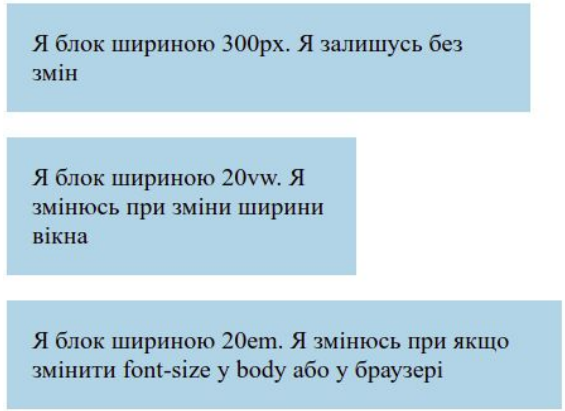
Відносні одиниці довжин є відносними до чогось ще, можливо до розміру батьківського шрифту або до розміру вікна перегляду. Перевага використання відносних одиниць полягає в тому, що ви можете зробити так, щоб розмір тексту або інших елементів масштабувались відносно всього іншого на сторінці.

Одиниця	Відносна до
`em`	Розмір шрифта **батьківського** елемента
`ex`	x-висота шрифта елемента
`ch`	Попередня міра (ширина) гліфа "0" шрифта елемента
`rem`	Розмір шрифта кореневого елемента
`lh`	Висота рядка елемента
`vw`	1% від ширини вікна перегляду
`vh`	1% від висоти вікна перегляду
`vmin`	1% від меншого значення ширини або висоти вікна перегляду
`vmax`	1% від більшого значення ширини або висоти вікна перегляду

Значення властивостей CSS: Приклад #1

```css
body {
    font-size: 1rem;      /* тобто 100% від базового розміру шрифта браузера, що для 97% користувачів згідно дослідження становить 16px */
}
.box {
    background-color: lightblue;
    padding: 1em;
    margin: 1em 0;
}
.px {
    width: 300px;
}
.vw {
    width: 20vw;
}
.em {
    width: 20em;
}
```

```html
<div class="wrapper">
    <div class="box px">Я блок шириною 300px. Я залишусь без змін</div>
    <div class="box vw">Я блок шириною 20vw. Я змінюсь при зміни ширини вікна</div>
    <div class="box em">Я блок шириною 20em. Я змінюсь при якщо змінити font-size у body або у браузері</div>
</div>
```



Переглянути у JSFiddle

Значення властивостей CSS: Приклад #2

```css
body {
    font-size: 2rem;      /* тобто 200% від базового
розміру шрифта браузера, що для 97% користувачів
згідно дослідження становить 16px (16px*2=32px) */
}
h1 {
    font-size: 1.5em;     /* тобто 150% від body */
}
.one {
    font-size: 1em;       /* тобто 100% від body */
}
.two {
    font-size: 0.5em;     /* тобто 50% від .one */
}
.bigger {
    font-size: 2em;       /* тобто 200% від .two */
}
```

```html
<h1>Текст1</h1>
<div class="one">
    Текст2
    <div class="two">
        Текст3
        <div class="bigger">
            Текст4
        </div>
    </div>
</div>
```

Текст1

Текст2

Текст3

Текст4

Переглянути у JSFiddle

Значення властивостей CSS: Колір

Ключові слова для задання кольору

Існує певна кількість ключових слів для задання кольору.
Наприклад: black, white, cyan, yellow, azure, coral, darkorange, indigo, lightpink, magenta, mintcream, royalblue, yellowgreen та багато інших.

Повний список ви можете подивитися на сторінці:
https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

Шістнадцяткова система числення

Шістнадцяткова система числення (hexadecimal, hex) — це позиційна система числення з основою 16.
Тобто кожне число в ній записується за допомогою 16 символів.
Арабські цифри від 0 до 9 відповідають значенням від нуля до дев'яти, а 6 літер латинської абетки (A, B, C, D, E, F) відповідають значенням від десяти до п'ятнадцяти.

Кожна шістнадцяткова цифра представляється чотирма бінарними цифрами (бітами), і основне застосування шістнадцяткового запису — це зручний запис двійкового коду.
Одна шістнадцяткова цифра є ніблом, який є половиною з октету або байту (8 біт).
Наприклад, значення байт лежить в діапазоні від 0 до 255 (в десяткових числах), але може бути більш зручно представити у вигляді двох шістнадцяткових цифр в діапазоні від 00 до FF.

Шістнадцяткова система широко використовується програмістами та для представлення адресації пам'яті комп'ютера.

Hex(16)	Dec(10)	Oct(8)	Bin(2)
0	0	0	0 0 0 0
1	1	1	0 0 0 1
2	2	2	0 0 1 0
3	3	3	0 0 1 1
4	4	4	0 1 0 0
5	5	5	0 1 0 1
6	6	6	0 1 1 0
7	7	7	0 1 1 1
8	8	10	1 0 0 0
9	9	11	1 0 0 1
A	10	12	1 0 1 0
B	11	13	1 0 1 1
C	12	14	1 1 0 0
D	13	15	1 1 0 1
E	14	16	1 1 1 0
F	15	17	1 1 1 1

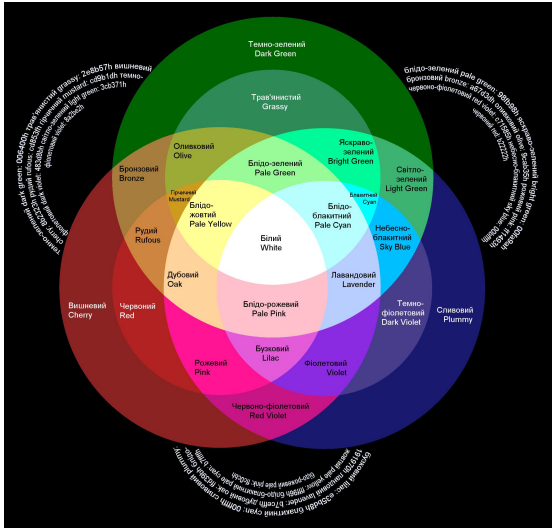
RGB

RGB (скорочено від Red, Green, Blue — червоний, зелений, синій) — колірна модель, що описує спосіб синтезу кольору, за якою червоне, зелене та синє світло накладаються разом, змішуючись у різноманітні кольори.

У даній моделі колір кодується градаціями складових каналів (**R**ed, **G**reen, **B**lue). Тому за збільшення величини градації котрогось каналу — зростає його інтенсивність під час синтезу.

Кількість градацій кожного каналу залежить від розрядності бітового значення RGB.
Зазвичай використовують 24-бітну модель, у котрій визначається по 8 біт на кожен канал, і тому кількість градацій дорівнює 256, що дозволяє закодувати 256^3 = 16 777 216 кольорів.

Детальніше:
https://uk.wikipedia.org/wiki/RGB



rtacademy.net/v3/03.rgb-colors.png

Значення властивостей CSS: Колір

Шістнадцяткові RGB значення

Кожне hex-значення складається з символу решітки (#) за яким йдуть 6 шістнадцяткових чисел, кожне з яких може приймати одне з 16 значень від 0 до f - тобто 0123456789abcdef.
Кожна пара значень являє один з каналів - червоного, зеленого або синього кольорів - і дозволяє нам визначати будь-який з 256 доступних значень для кожного (16 x 16 = 256).

Ці значення є трохи більш складними для розуміння, але вони набагато більш універсальні ніж ключові слова - ви можете використовувати hex-значення для відображення будь-якого кольору (особливо якщо він відсутній в таблиці зарезервованих кольорів).

```
.one {
    background-color: #02798b;  /* 0x02 => 2 червоного, 0x79 => 121 зеленого, 0x8b => 139 синього */
}
.two {
    background-color: #c55da1;  /* 0xc5 => 197 червоного, 0x5d => 93 зеленого, 0xa1 => 161 синього */
}
.three {
    background-color: #c55da180;  /* тут останній "80" - це альфа-канал прозорості в 50% або 0.5 */
}
```

Значення властивостей CSS: Колір

Функції rgb() та rgba()

Можливе використання функції rgb(), якій дається три параметра що представляють канали червоного, зеленого і синього значень кольорів, так само як і для hex-значень. Відмінність з використанням функції rgb() є те, що кожен канал представлений не двома hex-цифрами, а десятковим числом між 0 і 255.

rgba() — працює в точності, як і rgb(), однак існує четверте значення, яке представляє альфа канал кольору, яке контролює непрозорість, що приймає значення від 0 (прозоре) до 1 (непрозоре).

```
.one {
    background-color: rgb(2, 121, 139);
}
.two {
    background-color: rgba(197, 93, 161, 0.7);  /* 0.7 => непрозоре на 70%, а прозоре на 30% */
}
```

Стилізація тексту

Колір

Властивість color встановлює колір вмісту переднього плану обраних елементів.

```
p {
    color: #6495ed;
}
```

Родина шрифтів

Щоб встановити інший шрифт для вашого тексту необхідно використовувати властивість font-family, що дозволяє вам вказати шрифт (або список шрифтів), який браузер буде застосовувати до обраних елементів. Браузер буде застосовувати шрифт тільки в тому випадку, якщо він доступний на *комп'ютері клієнта*.
Оскільки ви не можете гарантувати доступність шрифтів, ви можете надати стек шрифтів, щоб браузер мав кілька шрифтів, з яких він може обирати (у вказаному порядку).

```
body {
    font-family: "Trebuchet MS", Verdana, sans-serif;
}
```

Стилізація тексту

Розмір шрифту

Встановлюється за допомогою властивості font-size

```
p {
    font-size: 1.25em;
}
```

Вирівнювання тексту

Властивість text-align використовується для керування вирівнюванням тексту всередині батьківського елемента. Можливі значення: left, right, center, justify (за шириною).

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

Стилізація тексту

Висота лінії (міжрядковий інтервал)

Властивість line-height встановлює висоту кожного рядка тексту, що може приймати більшість одиниць довжини і розміру, але також може приймати значення без одиниць виміру, яка діє як множник і зазвичай вважається кращим варіантом - розмір шрифту множиться щоб отримати висоту рядка.

```
.one {
    line-height: 1.5em;
}
.two {
    font-size: 1em;
    line-height: 3;     /* тобто буде 1em * 3 = 3em */
}
```

Стилізація тексту

Стиль шрифту

Властивість font-style використовується для включення і виключення курсивного тексту.
Можливі значення: normal, italic (курсив), oblique (коса версія шрифта, за відсутності - курсив)

```
p {
    font-style: italic;
}
```

Вага шрифту

Властивість font-weight встановлює, наскільки жирним є текст.
Можливі значення: normal, bold, lighter, bolder, 000-900

```
p {
    font-weight: bold;
}
```

Стилізація тексту

Перетворення тексту

text-transform перетворює текст.

Можливі значення: none (без змін), uppercase (верхній регістр), lowercase (нижній регістр), capitalize (перша буква слова у верхньому регістрі, інші в нижньому), full-width (для вирівнювання символа всередині квадрата)

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

Оформлення тексту

text-decoration встановлює/скасовує оформлення тексту.

Можливі значення: none (відсутнє), underline (підкресленний), overline (лінія над текстом), line-through (закресленний)

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

Стилізація посилань: Стан

Перше, що потрібно зрозуміти, це концепція станів посилань - різні стани, в яких можуть існувати посилання, що можуть бути стилізовані з використанням різних **псевдокласів**:

- **link**: стан за замовчуванням, в якому знаходиться посилання, коли воно не перебуває ні в якому іншому стані.
- **visited**: зовнішній вигляд посилання, коли воно вже переглянуло (існує в історії браузера)
- **hover**: зовнішній вигляд посилання, коли воно знаходиться під курсором миші
- **focus**: зовнішній вигляд посилання, в момент фокусу на ній
- **active**: зовнішній вигляд посилання, в момент активації (наприклад, в момент натискання)

Стилізація посилань

Порожній набір правил в правильному порядку.

Цей порядок важливий, тому що стилі посилань засновані один на одному. Наприклад, стилі в першому правилі будуть застосовуватися до всіх наступних, і коли посилання активується (:active), вона також має стан :hover.

Якщо ви поставите їх в неправильному порядку, все буде працювати неправильно.

```
a {
}

a:link {
}

a:visited {
}

a:focus {
}

a:hover {
}

a:active {
}
```

Довідник CSS

Довідник всіх стандартних властивостей, псевдокласів і псевдоелементів, @rules, одиниць виміру і селекторів CSS в алфавітному порядку:

https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

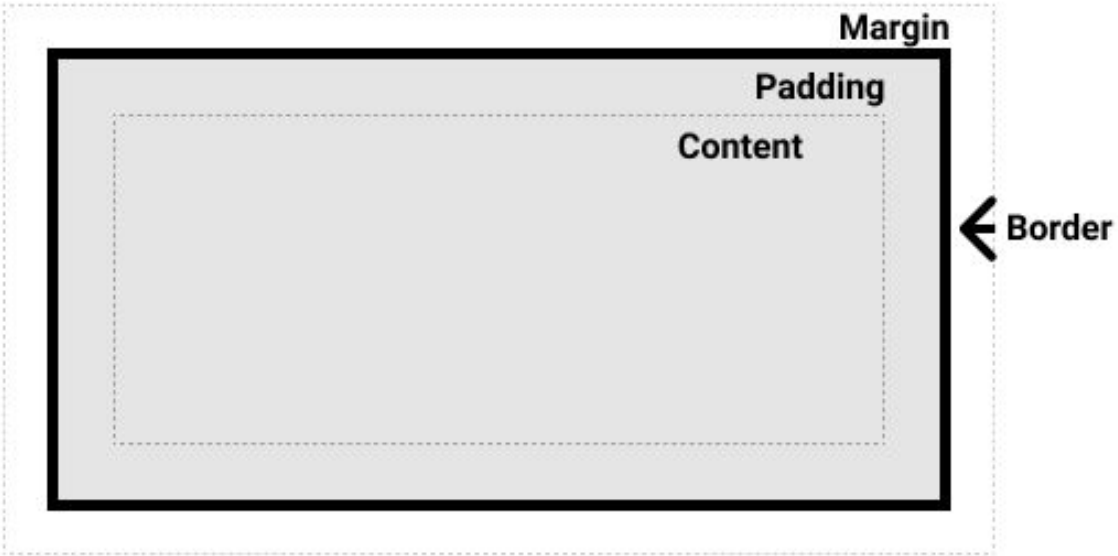
Завдання #3.2

1. Додайте правила в вашу таблицю стилів (styles.css), щоб
   - Родина шрифтів для всіх елементів, по замовчуванню, має бути Arial, а у випадку відсутності - будь-який з сімейства sans-serif
   - Розмір шрифта для всіх елементів має бути 16px
   - Міжрядковий інтервал для всіх елементів має бути 120%, а для h1-h6 - 140%
   - Колір шрифта для всіх елементів має бути #333333
   - Колір посилань має бути black (записати у HEX-вигляді)
   - Колір посилань при наведенні миші має бути darkblue (записати у HEX-вигляді) і не мати підкреслення
   - Всі посилання в елементі <nav> мають бути без підкреслення
   - Розташування <h1> має бути по-центру

2. Перевірте файл стилів на помилки за допомогою CSS-валідатора jigsaw.w3.org/css-validator

Блокова модель CSS

Блокова модель (боксова модель, box model) описує з чого складається блок і які властивості впливають на його розміри.
Модель визначає, як різні частини блока - margin (зовнішні відступи), border (рамка), padding (внутрішні відступи), і content (контент) працюють разом, щоб створити блок.



Блокова модель CSS: padding

Внутрішні поля елемента (padding) - це порожня область, що оточує контент.

Розміри полів задаються окремо з різних сторін властивостями padding-top, padding-right, padding-bottom, padding-left або загальною властивістю padding, записаною у вигляді короткого синтаксиса (shorthand).

Властивість padding може бути вказано з використанням одного, двох, трьох або чотирьох значень.
Кожне значення є <довжина> або <відсоток>. Негативні значення не допускаються.

1. Якщо задано одне значення, воно застосовує однакове заповнення до всіх чотирьох сторін.

2. Якщо задані два значення, перше заповнення застосовується до верху (top) і низу (bottom), друге - до лівого (left) і правого (right).

3. Якщо задані три значення, перший відступ застосовується до верхнього (top), другий - до лівого (left) і правого (right), третій - до нижнього (bottom).

4. Якщо вказано чотири значення, відступи застосовуються до верхнього (top), правого (right), нижнього (bottom) і лівого (left) значенням в цьому порядку (за годинниковою стрілкою від “12” до “9”).

Блокова модель CSS: padding: Приклад

```
.sample1 {
    padding: 10px;              /* top=10px, right=10px, bottom=10px, left=10px */
}

.sample2 {
    padding: 10px 0;            /* top=10px, right=0px, bottom=10px, left=0px */
}

.sample3 {
    padding: 10px 0 2em;        /* top=10px, right=0px, bottom=2em, left=0px */
}

.sample4 {
    padding: 10px 0 2em 5vw;    /* top=10px, right=0px, bottom=2em, left=5vw */
}
```

Блокова модель CSS: margin

Зовнішній відступ (margin) додає порожній простір навколо елемента і визначає відстань до сусідніх елементів.

Розмір відступів задається окремо в різних напрямках властивостями margin-top, margin-right, margin-bottom, margin-left або загальною властивістю margin, записаною у вигляді короткого синтаксиса (shorthand).

Властивість margin може бути вказано з використанням одного, двох, трьох або чотирьох значень.
Кожне значення є <довжина> або <відсоток>. Негативні значення не допускаються.

Правила такі ж як і для властивості padding.

Блокова модель CSS: border

Рамка (border) оточує поля елемента.

Властивість border описує не тільки ширину рамки, а і визначає її стиль, бо фактично ця властивість є поєднаннями border-color, border-style, border-width записаною у вигляді короткого синтаксиса (shorthand).

Але варто зазначити що border-color, border-style, border-width є також короткими синтаксисами.

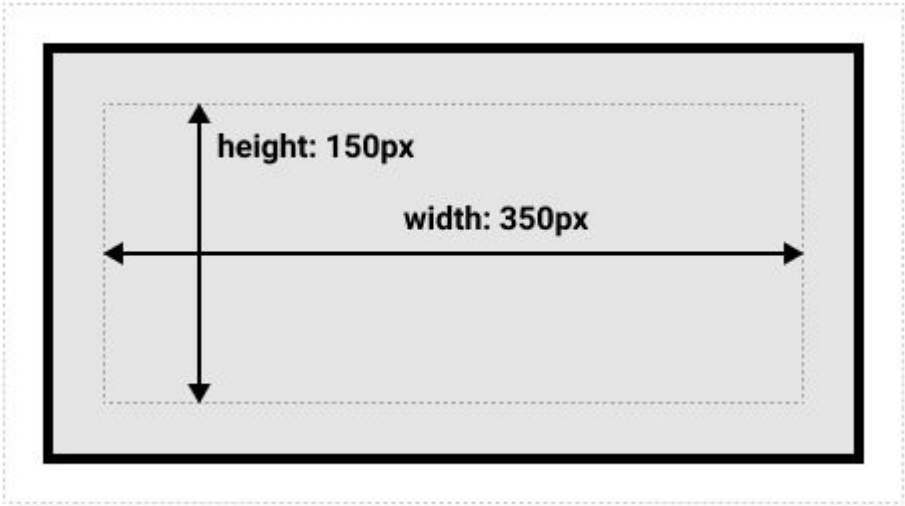
```
.sample1 {
    border: 10px solid #001122;
}

.sample2 {          /* стиль .sample2 такий же як і .sample1 */
    border-width: 10px;
    border-style: solid;
    border-color: #001122;
}
```

Стандартна блокова модель CSS

У стандартній блоковій моделі, вказуючи ширину (width) і висоту (height) для блока, це визначає ширину і висоту вмісту блока.
Будь-які відступи (margin, padding) і рамки (border) потім додаються до цієї ширини і висоти, щоб отримати загальний розмір, займаний блоком. Це показано на малюнку нижче.

```
.box {
    width: 350px;
    height: 150px;
    margin: 10px;
    padding: 25px;
    border: 5px solid #000000;
}
```

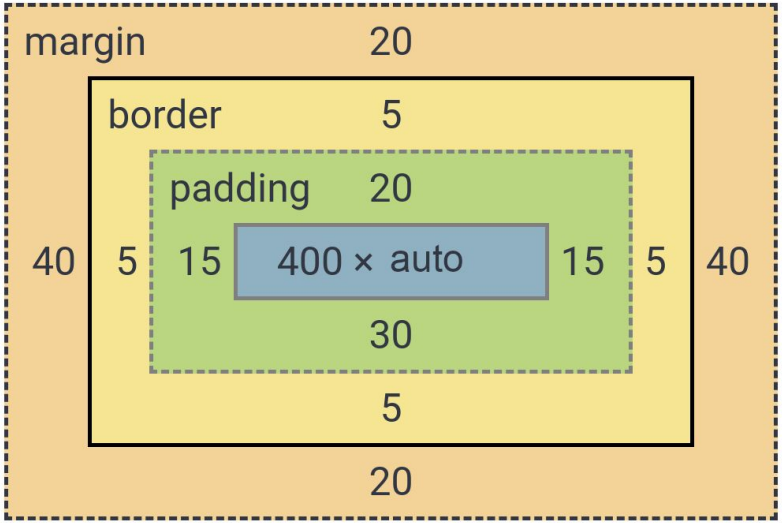
Тобто загальна ширина такого блока буде становити 430px (=10+5+25+350+25+5+10), а висота - 230px (=10+5+25+150+25+5+10).



Завдання #3.3

1. На окремій сторінці створіть блок шириною 400px, автоматичною висотою, кольором фону #f1faee, кольором границі #aad9dc, кольором шрифта #457b9d та шрифтом Arial.

   Також з відступами та границями як зображено на схематичному малюнку праворуч



2. Візуально блок з п.1 має бути як на малюнку праворуч

@rules: import, charset

@rules - це особливі правила, що дають CSS інструкції, як поводитися. У деяких правил @rules прості назви і значення.

@import використовується для імпорту ще однієї таблиці стилів всередині CSS-файлу.
Так браузер довантажить ще одну таблицю стилів одразу як побачить це правило. Але краще під'єднувати через ще один елемент <link>.

```
@import "other_styles.css";
```

@charset застосовується для завдання кодування зовнішнього CSS-файлу. Це має значення в тому випадку, якщо в CSS-файлі використовуються символи національного алфавіту.

```
@charset "utf-8";
```

@rules: font-face

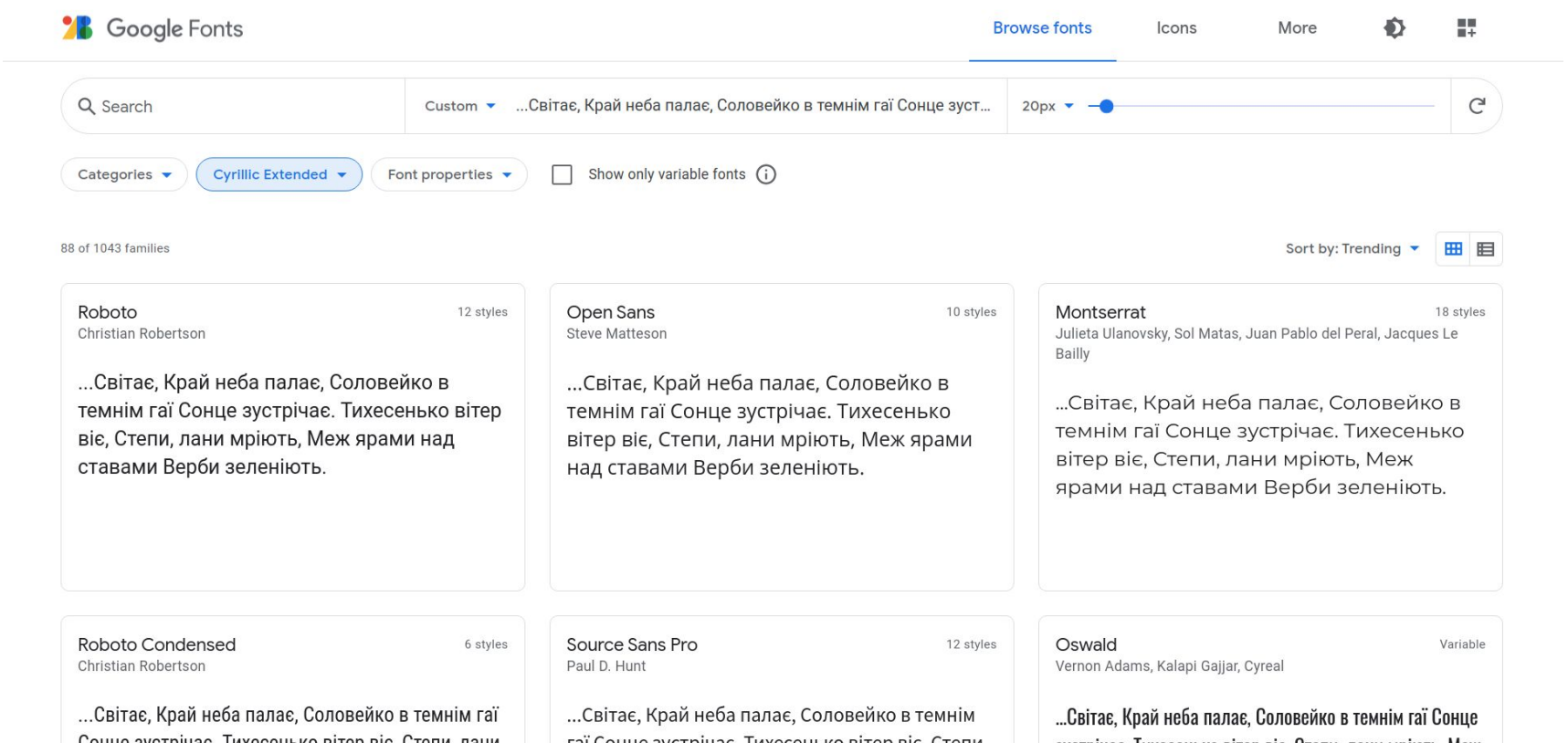
@font-face дозволяє вказати додаткові шрифти для відображення тексту на веб-сторінках, які можуть бути завантажені або з віддаленого сервера (за URL), або з комп'ютера користувача (local).

@font-face не може бути оголошений всередині CSS-селектора.
Список доступних форматів шрифта: "woff", "woff2", "truetype", "opentype", "embedded-opentype" и "svg".

```
@font-face {
    font-family: "Open Sans";
    src: url("/fonts/OpenSans-Regular-webfont.woff2") format("woff2"),
         url("/fonts/OpenSans-Regular-webfont.woff") format("woff");
}

body {
    font-family: "Open Sans", sans-serif;
}
```

Каталог шрифтів Google Fonts

https://fonts.google.com/



@rules: keyframes

@keyframes керує проміжними кроками в послідовності анімації CSS, визначаючи стилі для ключових кадрів (або шляхових точок) впродовж послідовності анімації.

Анімація складається з двох компонентів - CSS-стилю, що описує анімацію, і набору ключових кадрів, що вказують на початковий і кінцевий стан стилю анімації, а також можливих проміжних точок.

Щоб використовувати ключові кадри, створіть правило @keyframes з іменем, яке потім використовується властивістю анімації-імені для узгодження анімації з її ключовим кадром.
Кожне правило @keyframes містить список ключових кадрів, які визначають відсотки вздовж анімації, коли відбувається ключовий кадр, і блок, що містить стилі для цього ключового кадру.

Детальніше:
https://developer.mozilla.org/en-US/docs/Web/CSS/animation
https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations

Приклад лоадерів на CSS:
https://freefrontend.com/css-loaders/

@rules: keyframes: Приклад #1

```html
<div class="colors"></div>
```

```css
.colors {
    height: 5em;
    background-color: #264653;
    animation: color1 10s linear infinite;
    /* animation: <@keyframes-name> <duration>
                  <easing-function> <delay> */
}

@keyframes color1 {
    0%   { background-color: #264653; }
    20%  { background-color: #2a9d8f; }
    40%  { background-color: #e9c46a; }
    60%  { background-color: #f4a261; }
    80%  { background-color: #e76f51; }
}
```

Переглянути у JSFiddle

@rules: keyframes: Приклад #2

```html
<div class="loader"></div>
```

```css
.loader {
    width: 2em;
    height: 2em;
    border: 0.25em solid #ffffff;
    border-top: 0.25em solid #a0a0a0;
    border-radius: 50%;
    animation: spin 2s linear infinite;
    /* animation: <@keyframes-name> <duration>
                  <easing-function> <delay> */
}
@keyframes spin {
    0%    { transform: rotate(0deg); }
    50%   { transform: rotate(180deg); }
    100%  { transform: rotate(360deg); }
}
```

Переглянути у JSFiddle

@rules: media

Найчастіше зустрічаються @rules під назвою @media, що дозволяють використовувати media-запити для застосування CSS в певних випадках, тільки якщо виконуються ті чи інші умови (наприклад, при зміні розмірів вікна або при перегляді сайту з іншого типу пристрою).
https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

```
body {              /* ці стилі будуть застосовуватися за замовчуванням */
    background-color: white;
}
@media only screen and (min-width: 320px) and (max-width: 640px) {
                  /* ці стилі будуть застосовуватися тільки для звичайних екранів пристроїв,
                     у яких ширина від 320px до 640px */
    body {
        background-color: grey;
    }
}
@media print {     /* ці стилі будуть застосовуватися тільки для версії для друку */
    img {
        display: none;
    }
}
```

Media types (типи пристроїв)

Типи пристроїв описують загальну категорію пристроїв, з яких проглядається сторінка.
Існують наступні типи пристроїв:

- **screen** - призначений для екранів
- **print** - призначений для сторінок і документів, що переглядаються на екрані в режимі попереднього перегляду друку та при друку
- **speech** - призначений для мовних синтезаторів (що використовуються для незрячих користувачів)
- **all** - підходить для всіх пристроїв

Тип пристроїв можна вказувати як в media-запитах, так і атрибутом media для тега <link> при підключенні зовнішньої таблиці стилів:

```
<link rel="stylesheet" media="print" href="/css/print.css">
```

Детальніше:
https://www.sitepoint.com/css-printer-friendly-pages/
https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

Viewport

Viewport - це видима користувачу область веб-сторінки, що він бачить не вдаючись до прокручування.

Meta-тег viewport повідомляє браузеру про те, як саме обробляти розміри сторінки і змінювати її масштаб. Цей тег необхідно додавати в секцію `<head>`:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

де

- `width` - ширина області перегляду.
  Значення `device-width` задає ширину сторінки відповідно до розміру екрана.

- `initial-scale` - початковий масштаб сторінки.
  Значенням атрибута є числом від 0.1 до 1.0.
  Значення 1.0 визначає масштаб 1:1, тобто «Не масштабувати».

Завдання #3.4

1. Для блока з завдання #3.3 встановіть наступні стилі:

```
div {
    width: 640px;
    margin: 0;
    padding: 40px;
    border: none;
    background-color: #457b9d;
    color: #ffffff;
    font-family: Arial, sans-serif;
}
```

2. Додайте мета-тег viewport

3. Використовуючи media-запити необхідно змінювати фон цього блока в залежності від ширини екрана (перевіряти краще за допомогою емуляції пристроїв у Chrome DevTools: F12 -> Ctrl+Shift+M):

   a. Починаючи з 768px колір фона має бути #e9c46a (Orange Yellow Crayola)

   b. З 992px колір фона має бути #f4a261 (Sandy Brown)

   c. З 1200px і більше - #e76f51 (Burnt Sienna)

Види верстки сторінок

- Фіксована (Fixed Layout, рос. “Фиксированная вёрстка”)
- Еластична (Elastic layout, рос. “Резиновая вёрстка”)
- Адаптивна (Adaptive Layout, рос. “Адаптивная вёрстка”)
- Чутлива (Responsive Layout, рос. “Отзывчивая вёрстка”)

Види верстки сторінок: Фіксована

Фіксована верстка (Fixed Layout, рос. “Фиксированная вёрстка”) - підхід створення сторінок сайту, які мають задану ширину. Ширина компонентів на сторінці не змінюється.

На моніторах з маленькою роздільною здатністю з'являється горизонтальна смуга прокрутки.
Даний тип верстки не підходить для зручного відображення інформації на мобільних пристроях.

Приклад нижче демонструє строге завдання ширини для тега в 1200px незалежно від розміру екрана:

```
div {
    width: 1200px;      /* ширина блока завжди буде становити 1200px */
}
```

Види верстки сторінок: Еластична

Еластична верстка (Elastic layout, рос. “Резиновая вёрстка”) має на увазі можливість компонентів сайту міняти свої розміри в залежності від розміру вікна браузера - розтягуватися від і до зазначених мінімальних і максимальних розмірів.

Це досягається завдяки використанню відносних значень, max-width/min-width (максимальна / мінімальна ширина), max-height/min-height (максимальна / мінімальна висота).

Приклади використання технік еластичної верстки:

```
.sample1 {
    width: 90%;          /* ширина блока буде становити 90% відносно ширини батіківського елементу */
}

.sample2 {
    min-width: 360px;    /* ширина блока не буде зменшуватися після 360px */
    max-width: 1200px;   /* ширина блока не буде збільшуватися після 1200px */
}
```

Види верстки сторінок: Адаптивна

Адаптивна верстка (Adaptive Layout, рос. “Адаптивная вёрстка”) дозволяє підлаштовуватися основному контейнеру і будь-якому іншому елементу сайту під роздільну здатність екрану, роблячи можливим змінювати розмір шрифту, розташування об'єктів, колір і т.п.

Це відбувається динамічно за використанням медіа-запитів (@media), що дозволяють автоматично визначати роздільну здатність екрану, тип пристрою і підставляти зазначені значення в автоматичному режимі.

У прикладі нижче ширина буде становити 1140px для всіх пристроїв, ширина екрану яких менше або дорівнює 1200px. А при ширині екрану менше або дорівнює 576px - ширина буде становити 540px.

```
@media (max-width: 1200px) {
    div {
        width: 1140px;    /* ширина блока буде 1140px при розмірі екрана <= 1200px */
    }
}
@media (max-width: 576px) {
    div {
        width: 540px;     /* ширина блока буде 540px при розмірі екрана <= 576px */
    }
}
```

Види верстки сторінок: Чутлива

Чутлива верстка (Responsive Layout, рос. “Отзывчивая вёрстка”) - це об'єднання еластичної і адаптивної верстки.

При цьому підході використовуються як медіа-запити, так і процентне завдання ширини компонентів. Використовуючи даний вид верстки можна з упевненістю сказати, що сайт пристосується до будь-якого пристрою.

Нижче задається ширина рівна 50% від розміру батьківського елемента для всіх пристроїв, ширина яких менше 75em (або 1200px). А для пристроїв ширина яких менше 36em (або 576px) - ширина буде становити 100%.

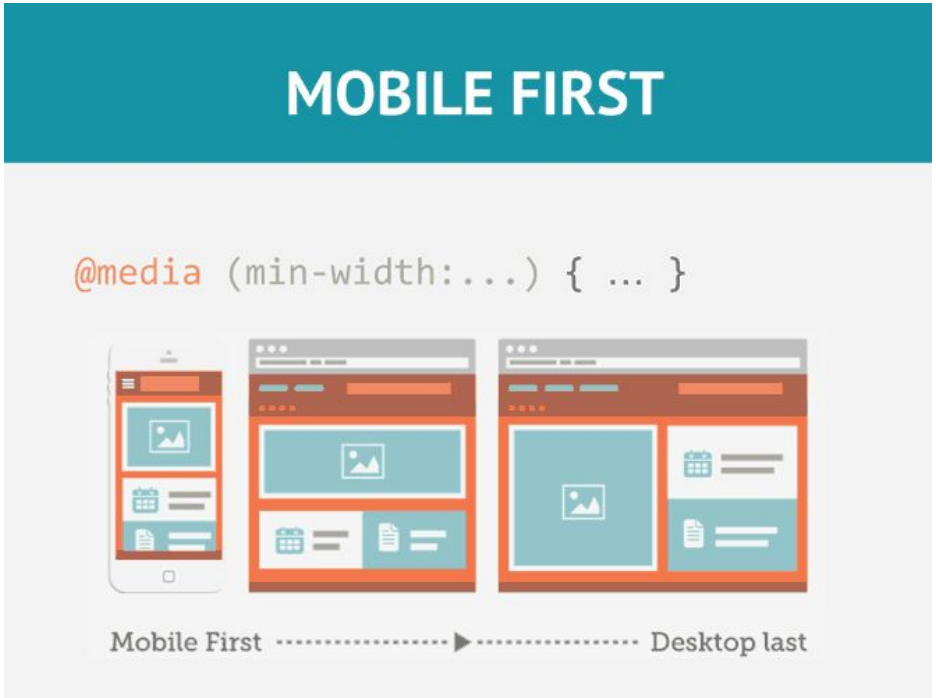
```
@media (max-width: 75em) {
    div {
        width: 50%;     /* ширина блока буде 50% при розмірі екрана <= 75em або 1200px */
    }
}
@media (max-width: 36em) {
    div {
        width: 100%;    /* ширина блока буде 100% при розмірі екрана <= 36em або 576px*/
    }
}
```

Mobile First

Необхідно спочатку визначати стилі для мобільних телефонів (**Mobile First**) і потім додавати інші пристрої з більшою роздільною здатністю за допомогою окремих блоків з media-запитами, а не навпаки.

Поширені розміри екрану:

- Мобільний телефон: 360 x 640
- Мобільний телефон: 375 x 667
- Мобільний телефон: 360 x 720
- iPhone X: 375 x 812
- Pixel 2: 411 x 731
- Tablet (Планшет): 768 x 1024
- Laptop (Ноутбук): 1366 x 768
- Настільний комп'ютер (Desktop): 1920 x 1080



Media-запити у px

Приклад використання media-запитів для динамічного відображення блока відносно ширини екрана на основі значення у px, що використовується у CSS-фреймворку [Bootstrap v5.0]

```
<div class="container">
    <!-- тут вміст блока -->
</div>
```

```
.container {
    width: 100%;
    margin-right: auto;
    margin-left: auto;
}
@media (min-width: 576px) {
    .container {
        max-width: 540px;
    }
}
@media (min-width: 768px) {
    .container {
        max-width: 720px;
    }
}
@media (min-width: 992px) {
    .container {
        max-width: 960px;
    }
}
@media (min-width: 1200px) {
    .container {
        max-width: 1140px;
    }
}
@media (min-width: 1400px) {
    .container {
        max-width: 1320px;
    }
}
```

Media-запити у em

Рекомендується використовувати одиницю em (або rem) для media-запитів для коректного масштабування сторінки, оскільки при вказанні media-запитів у px масштабування відображається некоректно.

1em згідно дослідження для 97% користувачів становить 16px

Перетворені у em media-запити з CSS-фреймворку Bootstrap v5.0

```
<div class="container">
    <!-- тут вміст блока -->
</div>
```

```
.container {
    width: 100%;
    margin-right: auto;
    margin-left: auto;
}
@media (min-width: 36rem) {
    .container {
        max-width: 34rem;
    }
}
@media (min-width: 48rem) {
    .container {
        max-width: 45rem;
    }
}
@media (min-width: 62rem) {
    .container {
        max-width: 60rem;
    }
}
@media (min-width: 75rem) {
    .container {
        max-width: 71rem;
    }
}
@media (min-width: 88rem) {
    .container {
        max-width: 82rem;
    }
}
```

Завдання #3.5

1. Перетворіть значення у px в одиниці відносної довжини (em)

2. Додайте підтримку media-запитів для чутливої (responsive) верстки (у відносних одиницях - em).

3. Перевірте за допомогою інструменту від Гугл search.google.com/test/mobile-friendly *(у цьому випадку - вставляючи код, а не адресу посилання на файл)*

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <style>
        main {
            font: 20px Arial, sans-serif;
            min-width: 160px;
            width: 100%;
            margin: 0 auto;
            padding: 0;
            background-color: #457b9d;
        }
        .container {
            font-size: 32px;
            padding: 30px 64px 64px;
        }
        .container h1 {
            font-size: 40px;
        }
    </style>
</head>
<body>
    <main>
        <div class="container">
            <h1>Тестовий контент H1</h1>
            Тестовий контент тестовий контент тестовий контент
        </div>
    </main>
</body>
</html>
```

Адаптивні зображення

Адаптивні зображення - зображення, які добре працюють на пристроях із дуже різними розмірами екрана, роздільною здатністю та іншими подібними функціями.
Це допомагає покращити продуктивність на різних пристроях.

https://developer.mozilla.org/ru/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images

Завдання #3.6

1. На окремій сторінці додайте два зображення за допомогою тегів <img> та <picture> (бажано >1024px за шириною)

2. Зробіть ці зображення адаптивними мінімум для трьох розмірів екрану/пристрою.

3. * Додайте окремі зображення у форматі .webp для елемента <picture>, що браузер має завантажувати в першу чергу

CSS верстка

Методи компонування сторінок CSS дозволяють нам контролювати позицію елементів розташованих на веб-сторінці, змінювати їх позицію за замовчуванням та відносно інших елементів або головного вікна.

Розрізняють наступні методи:

- Нормальний потік (Normal flow)
- Властивість display (The display property)
- Flexbox
- Grid
- Floats
- Позиціонування (Positioning)
- Макет таблиці (Table layout)
- Багатостовпчиковий макет (Multiple-column layout)

Кожен метод має свої переваги і недоліки і жодна техніка не призначена для використання в ізоляції від інших.

CSS верстка: Нормальний потік

Нормальний потік (Normal flow) це те як ваш браузер відображає контент за замовчуванням, коли ви не міняли розташування елементів на сторінці.

Наприклад:

```
<p>I love my cat.</p>
<ul>
  <li>Buy cat food</li>
  <li>Exercise</li>
  <li>Cheer up friend</li>
</ul>
<p>The end!</p>
```

Переглянути у JSFiddle

За замовчуванням ваш браузер виведе цей код наступним чином:

I love my cat.

- Buy cat food
- Exercise
- Cheer up friend

The end!

Зауважте, що HTML елементи тут відображаються точно в такому ж порядку, як і у вихідному коді - перший параграф, за ним невпорядкований список, потім другий параграф.

CSS верстка: Властивість display

Значення властивості display є головними методами верстки розмітки сторінки в CSS.
Ця властивість дозволяє нам змінювати спосіб відображення за замовчуванням.

Кожен елемент за замовчуванням має властивість display, що впливає на те, як цей елемент відображається. Наприклад, параграфи розташовуються один під іншим тільки тому що вони мають за замовчуванням властивість display: block.

Якщо ж ви створите посилання всередині параграфа, то воно буде відображатися в загальному потоці з іншим текстом, без перенесення на новий рядок. Це тому що у елемента <a> за замовчуванням встановлено властивість display: inline.

Ви можете змінити поведінку display.
Наприклад, <li> відображається як display: block за замовчуванням, що означає що елементи списку відображаються один під іншим в нашому документі. Якщо ми змінимо значення display на inline вони будуть відображатися один за одним в ряд, як це роблять слова в реченні.

Той факт, що ви можете змінити значення display для будь-якого елемента означає, що ви можете обирати HTML елементи по їх семантичному значенню, не турбуючись про те як вони будуть відображатись.

CSS верстка: Властивість display: inline-block

Існує спеціальне значення display, яке забезпечує золоту середину між рядком і блоком - це inline-block.

Це корисно в ситуаціях, коли ви не хочете, щоб елемент переносився на новий рядок, але хочете, щоб він враховував ширину і висоту і уникав перекриття.

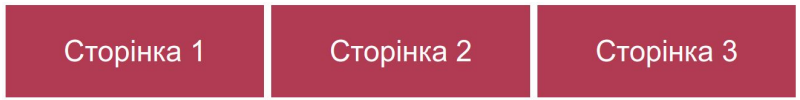
При цьому властивості width і height враховуються, як і відступи (padding, margin) та рамки/границі (border).

```
span {
    display: inline-block;
    margin: 2em;
    padding: 1em;
    width: 20em;
    height: 5em;
    border: 0.1em solid #0645ad;
}
```

CSS верстка: Властивість display: inline-block

Приклад використання display: inline-block для посилань у вигляді списка в блоці <nav>, що будуть відображатись один за одним в один ряд.

Для коректного відображення рекомендується вкладати елементи з display: inline-block в елемент з display: block




```
<nav>
    <ul class="links">
        <li><a href="/page1.html">Сторінка 1</a></li>
        <li><a href="/page2.html">Сторінка 2</a></li>
        <li><a href="/page3.html">Сторінка 3</a></li>
    </ul>
</nav>
```

```
.links li {
    background-color: #b33951;
    padding: 1em 2em;
    display: inline-block;
}
.links li a {
    color: #ffffff;
    text-decoration: none;
    font-family: Arial, sans-serif;
}
```

Переглянути у JSFiddle

CSS верстка: Flexbox

Flexbox (скорочення від Flexible Box Layout) - це модуль, розроблений для полегшення верстки в одному з вимірів - як рядок або стовпчик.

Для використання, встановіть властивість
display: flex
для **батьківського** елемента тих елементів, до яких потрібно застосувати цей тип верстки.
Всі його прямі нащадки стануть flex елементами.


```
<div class="wrapper">
    <div class="box1">Один</div>
    <div class="box2">Два</div>
    <div class="box3">Три</div>
</div>
```

```
.wrapper div {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.1em;
}
.wrapper {
    display: flex;
}
```

Переглянути у JSFiddle

CSS верстка: Flexbox

Існує властивість flex, що застосовуються до **вкладених** елементів. З її допомогою можна змінювати розміри елемента, розтягуючи його і примушуючи займати все доступне місце.

Властивість flex є скороченням (shorthand) від окремих властивостей flex-grow, flex-shrink і flex-basis.


```
<div class="wrapper">
    <div class="box1">Один</div>
    <div class="box2">Два</div>
    <div class="box3">Три</div>
</div>
```

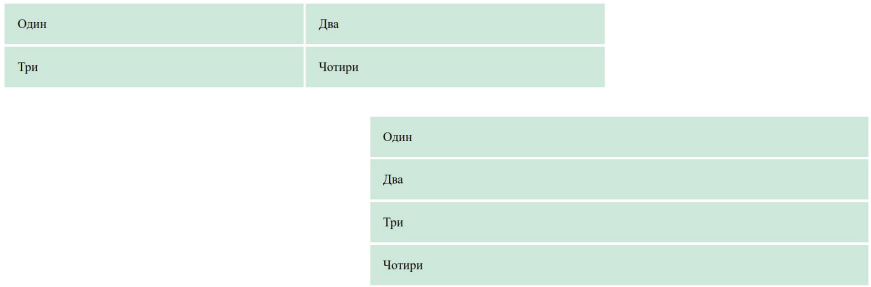
```
.wrapper div {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.1em;
    flex: 1 0 auto;
    /* flex: <flex-grow> <flex-shrink> <flex-basis> */
}
.wrapper {
    display: flex;
}
```

Переглянути у JSFiddle

CSS верстка: Flexbox

Властивість CSS flex-wrap для **батьківського** елемента задає правила виведення flex-елементів - в один рядок або в кілька, з переносом блоків.

Якщо перенесення дозволено, то можна задати напрямок, в якому виводяться блоки.
За замовчуванням перенос не дозволений (значення nowrap). При встановленні значення wrap вмикається можливість розташування елементів в кілька ліній (при менших розмірах екрану).



Переглянути у JSFiddle

```
<div class="wrapper">
    <div class="box1">Один</div>
    <div class="box2">Два</div>
    <div class="box3">Три</div>
    <div class="box4">Чотири</div>
</div>
```

```
.wrapper div {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.1em;
    flex: 1 0 auto;
    width: 45%;     /* задаємо розмір елемента */
}
.wrapper {
    display: flex;
    flex-wrap: wrap;
}
```

CSS верстка: Flexbox

Як працює CSS Flexbox. Наочне введення в систему компонування елементів на веб-сторінці.

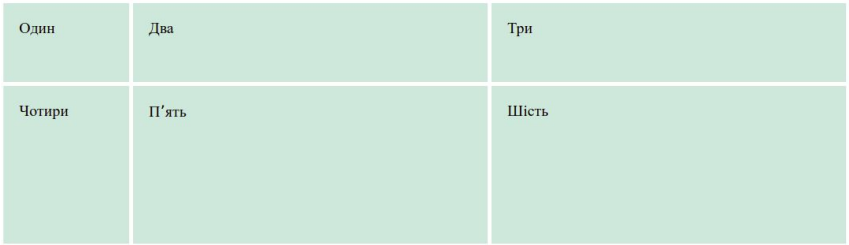
https://www.freecodecamp.org/news/flexbox-the-ultimate-css-flex-cheatsheet/

https://www.freecodecamp.org/news/an-animated-guide-to-flexbox-d280cf6afc35/

CSS верстка: Grid

У той час як Flexbox призначений для одновимірної розмітки, **Grid Layout** призначений для двовимірної - вибудовуючи предмети в ряди і стовпці.
Це робиться при встановленні display: grid.

Гнучка довжина або <flex> - це розмірність з одиницею виміру fr, яка представляє частку простору, що залишився в сітковому контейнері.



Переглянути у JSFiddle

```
<div class="wrapper">
    <div class="box1">Один</div>
    <div class="box2">Два</div>
    <div class="box3">Три</div>
    <div class="box4">Чотири</div>
    <div class="box5">П'ять</div>
    <div class="box6">Шість</div>
</div>
```

```
.wrapper {
    display: grid;
    grid-template-columns: 8em 1fr 1fr;
    grid-template-rows: 5em 10em;
    /* 5em - перший ряд, 10em - другий ряд */
    grid-gap: 0.25em; /* 0.25em - відступ між блоками */
}
.wrapper div {
    background-color: #cfe8dc;
    padding: 1em;
}
```

CSS верстка: Grid

Для випадку коли вам необхідно відобразити блоки у визначеній сітці, а не покладатися на поведінку авто-розміщення, необхідно використовувати властивості grid-row та grid-column для дочірніх елементів.

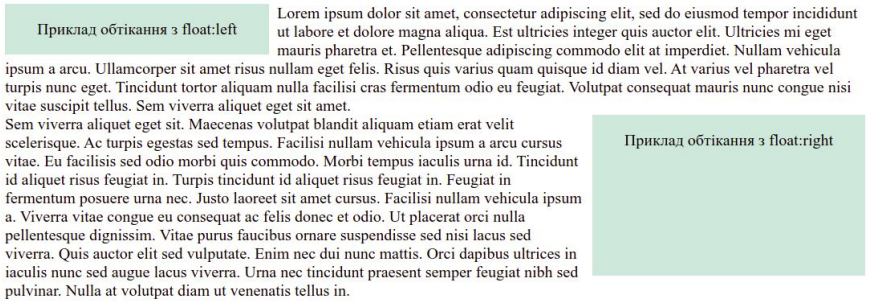
Переглянути у JSFiddle

```
.wrapper {
    display: grid;
    grid-template-columns: 8em 1fr 1fr;
    grid-template-rows: 5em 10em;
    grid-gap: 0.25em;
}
.box1 {
    grid-row: 1;
    grid-column: 2 / 4;
}
.box2 {
    grid-row: 1 / 3;
    grid-column: 1;
}
.box3 {
    grid-row: 2;
    grid-column: 3;
}
.box4 {
    grid-row: 3;
    grid-column: 1 / 4;
}
```

CSS верстка: Floats

Роблячи елемент плаваючим (float) ми міняємо поведінку цього елементу і елементів блочного рівня, наступних за ним в нормальному потоці.

Елемент переміститься вліво або вправо і видаляється з нормального потоку (normal flow), а навколишній контент оточує плаваючий елемент.



Переглянути у JSFiddle

Властивість float має чотири можливих значення:

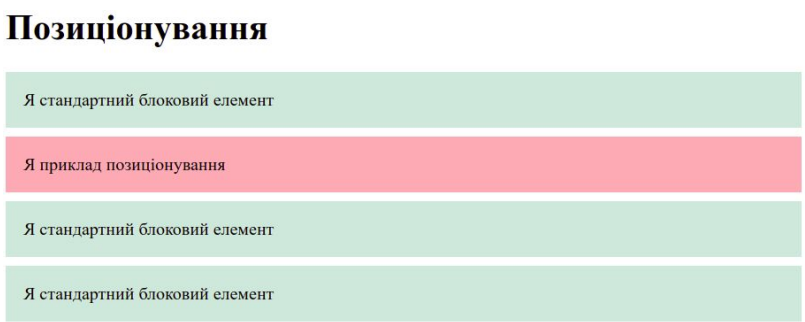
- left — Елемент вирівнюється зліва і інші елементи оточують його справа.
- right — Елемент вирівнюється праворуч і інші елементи оточують його зліва.
- none — Не застосовує float зовсім. Це значення за замовчуванням.
- inherit — Визначає, що значення властивості float має бути успадковано від батьківського елемента.

```
.float-left {
    float: left;
}
```

CSS верстка: Статичне позиціонування

Статичне позиціонування (Static positioning) - замовчування, яке отримують всі елементи, тобто елемент в їх "нормальній" позиції.

```
<h1>Позиціонування</h1>
<p>Я стандартний блоковий елемент</p>
<p class="positioned">Я приклад позиціонування</p>
<p>Я стандартний блоковий елемент</p>
```




```
p {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.5em 0 0 0;
}
.positioned {
    background-color: #ff546880;
    /* тут останній "80" - це альфа-канал прозорості
       в 50% або 0.5 */
}
```

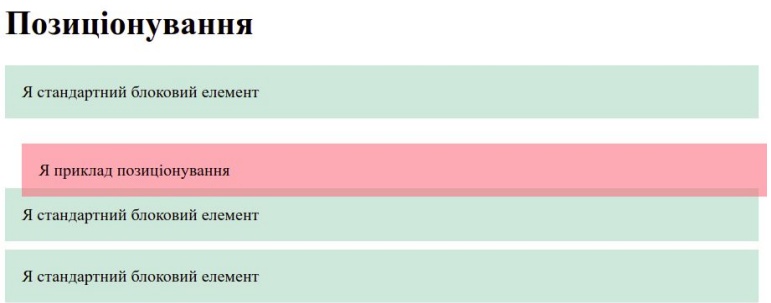
Переглянути у JSFiddle

CSS верстка: Відносне позиціонування

Відносне позиціонування (Relative positioning) дозволяє вам зміщувати елемент щодо положення, яке він би мав за замовчуванням в нормальному потоці.
Це означає, що ви можете виконати, наприклад, таку задачу як переміщення іконки трохи вниз, так щоб вона була на одній лінії з текстом.

```
<h1>Позиціонування</h1>
<p>Я стандартний блоковий елемент</p>
<p class="positioned">Я приклад позиціонування</p>
<p>Я стандартний блоковий елемент</p>
```

```
p {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.5em 0 0 0;
}
.positioned {
    background-color: #ff546880;
    position: relative;
    top: 1em;
    left: 1em;
}
```

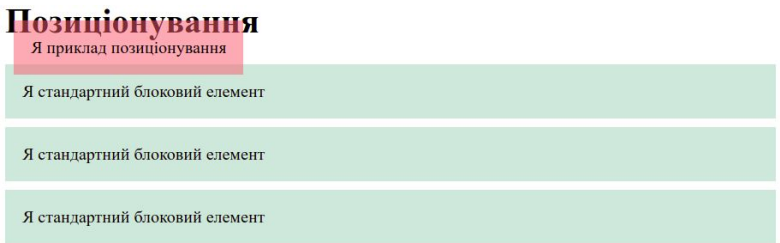


Переглянути у JSFiddle

CSS верстка: Абсолютне позиціонування

Абсолютне позиціонування (Absolute positioning) використовується щоб повністю видалити елемент з нормального потоку і розмістити його, використовуючи зміщення від країв батьківського елементу.

Позиційований елемент тепер зовсім відділений від розмітки сторінки і розташовується поверх всіх елементів.



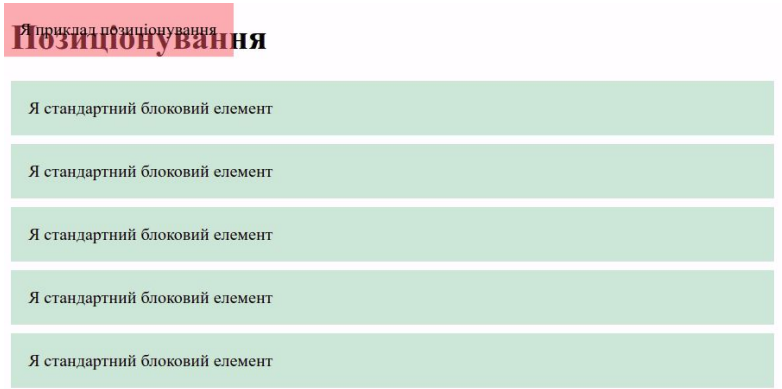

```
<h1>Позиціонування</h1>
<p>Я стандартний блоковий елемент</p>
<p class="positioned">Я приклад позиціонування</p>
<p>Я стандартний блоковий елемент</p>
```

```
p {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.5em 0 0 0;
}
.positioned {
    background-color: #ff546880;
    position: absolute;
    top: 2em;
    left: 1em;
}
```

Переглянути у JSFiddle

CSS верстка: Фіксоване позиціонування

Фіксоване позиціонування (Fixed positioning) видаляє елемент з нормального потоку документа так само, як і абсолютне позиціонування.
Однак, замість зсуву відносно контейнера, воно застосовується відносно вікна перегляду.
Так ми можемо створювати такі ефекти як меню, яке залишається зафіксованим поки сторінка прокручується під ним.



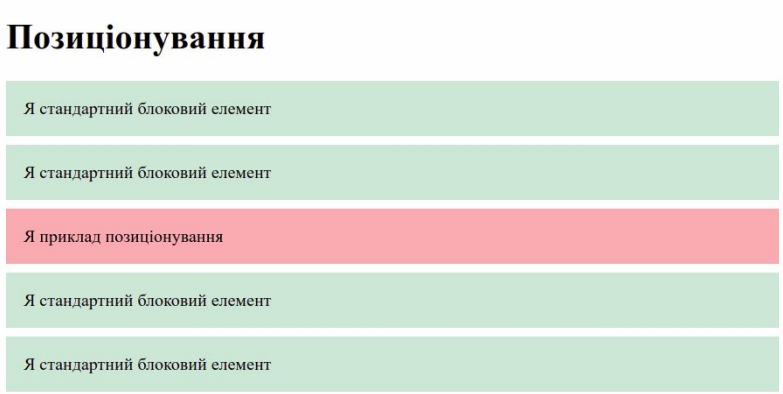
Переглянути у JSFiddle

```
<h1>Позиціонування</h1>
<p>Я стандартний блоковий елемент</p>
<p class="positioned">Я приклад позиціонування</p>
<p>Я стандартний блоковий елемент</p>
```

```
p {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.5em 0 0 0;
}
.positioned {
    background-color: #ff546880;
    position: fixed;
    top: 0;
    left: 0;
}
```

CSS верстка: Липке позиціонування

Липке позиціонування (Sticky positioning) - це мікс статичного позиціонування з фіксованим позиціонуванням.
Якщо елемент має position: sticky - він буде прокручуватися в нормальному потоці доки не досягне кордону вікна перегляду яке ми задали. Після чого елемент "прилипає", як якщо б був застосований position: fixed.



Переглянути у JSFiddle

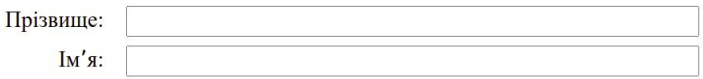
```html
<h1>Позиціонування</h1>
<p>Я стандартний блоковий елемент</p>
<p class="positioned">Я приклад позиціонування</p>
<p>Я стандартний блоковий елемент</p>
```

```css
p {
    background-color: #cfe8dc;
    padding: 1em;
    margin: 0.5em 0 0 0;
}
.positioned {
    background-color: #ff546880;
    position: sticky;
    top: 0;
    left: 0;
}
```

CSS верстка: Макет таблиці

Макет таблиці (Table layout) дозволяє застосовувати властивості таблиць та їх складових до елементів, що не є таблицями.

Встановлюється за допомогою display:table для батьківського елемента. Після чого необхідно задати елементи що будуть аналогами <tr> (ряду таблиці) - display:table-row та елементи що будуть аналогами <td> (комірками таблиці) - display:table-cell



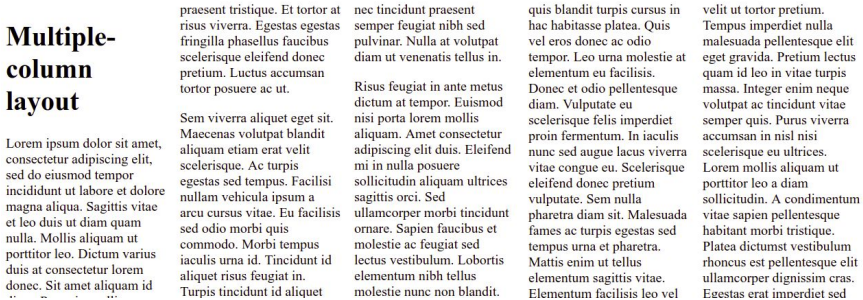
Переглянути у JSFiddle

```
<form>
    <div>
        <label>Прізвище:</label><input type="text">
    </div>
    <div>
        <label>Ім'я:</label><input type="text">
    </div>
</form>

form {
    display: table;
}
form div {
    display: table-row;
}
form label, form input {
    display: table-cell;
    width: 30em;
}
form label {
    width: 20em;
    padding-right: 1em;
    text-align: right;
}
```

CSS верстка: Багатостовпчиковий макет

Багатостовпчиковий макет (Multiple-column layout) дає нам змогу розташувати контент в шпальтах, подібно до того, як текст розташовується в газеті.

Щоб перетворити блок в такий контейнер ми використовуємо властивість column-count, що говорить браузеру скільки колонок ми хочемо мати або властивість column-width, що говорить браузеру заповнити контейнер якомога більшою кількістю стовпців такої ширини.



Multiple-column layout

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Sagittis vitae et leo duis ut diam quam nulla. Mollis aliquam ut porttitor leo. Dictum varius duis at consectetur lorem donec. Sit amet aliquam id

praesent tristique. Et tortor at risus viverra. Egestas egestas fringilla phasellus faucibus scelerisque eleifend donec pretium. Luctus accumsan tortor posuere ac ut.

Sem viverra aliquet eget sit. Maecenas volutpat blandit aliquam etiam erat velit scelerisque. Ac turpis egestas sed tempus. Facilisi nullam vehicula ipsum a arcu cursus vitae. Eu facilisis sed odio morbi quis commodo. Morbi tempus iaculis urna id. Tincidunt id aliquet risus feugiat in. Turpis tincidunt id aliquet

nec tincidunt praesent semper feugiat nibh sed pulvinar. Nulla at volutpat diam ut venenatis tellus in.

Risus feugiat in ante metus dictum at tempor. Euismod nisi porta lorem mollis aliquam. Amet consectetur adipiscing elit duis. Eleifend mi in nulla posuere sollicitudin aliquam ultrices sagittis orci. Sed ullamcorper morbi tincidunt ornare. Sapien faucibus et molestie ac feugiat sed lectus vestibulum. Lobortis elementum nibh tellus molestie nunc non blandit.

quis blandit turpis cursus in hac habitasse platea. Quis vel eros donec ac odio tempor. Leo urna molestie at elementum eu facilisis. Donec et odio pellentesque diam. Vulputate eu scelerisque felis imperdiet proin fermentum. In iaculis nunc sed augue lacus viverra vitae congue eu. Scelerisque eleifend donec pretium vulputate. Sem nulla pharetra diam sit. Malesuada fames ac turpis egestas sed tempus urna et pharetra. Mattis enim ut tellus elementum sagittis vitae. Elementum facilisis leo vel

velit ut tortor pretium. Tempus imperdiet nulla malesuada pellentesque elit eget gravida. Pretium lectus quam id leo in vitae turpis massa. Integer enim neque volutpat ac tincidunt vitae semper quis. Purus viverra accumsan in nisl nisi scelerisque eu ultrices. Lorem mollis aliquam ut porttitor leo a diam sollicitudin. A condimentum vitae sapien pellentesque habitant morbi tristique. Platea dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim cras. Egestas erat imperdiet sed

```
<div class="container">
    <h1>Multiple-column layout</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Sagittis vitae et leo duis ut diam quam nulla. Mollis aliquam ut porttitor leo. Dictum varius duis at consectetur lorem donec. Sit amet aliquam id diam. Purus in mollis…
    </p>
</div>
```

```
.container {
    column-width: 10em;
}
```

Переглянути у JSFiddle

Завдання #3.7

Спробуйте всі види компонування сторінок за допомогою CSS

Завдання #3.8

Оберіть будь-який дизайн за посиланням для вашого майбутнього блогу (*купляти не треба!*)

Зверстайте першу сторінку (*homepage*) вашого майбутнього блогу згідно обраного дизайну з наступними вимогами:

a. без використання фреймворку
b. чутлива (responsive) верстка, підтримка різних типів пристроїв та розмірів екрану
c. верстка Mobile First
d. підтримка версії для друку (print-версія)
e. всі значення в одиницях відносної довжини (%, em/rem, vw, vh і т.п.)
f. кольори у hex
g. адаптивні зображення
h. наявність структурованих даних (або мікроформати або схема JSON-LD)
i. проходить перевірку без помилок HTML та CSS-валідаторами
j. виглядає однаково для основних браузерів (Chrome, Safari, Edge, Firefox)
k. без вбудованих (inline) стилів

Завдання #3.9

Для обраного дизайну у завданні #3.8 зверстайте сторінку перегляду одного запису (*single*) вашого майбутнього блогу згідно обраного дизайну з наступними вимогами:

a. без використання фреймворку
b. чутлива (responsive) верстка, підтримка різних типів пристроїв та розмірів екрану
c. верстка Mobile First
d. підтримка версії для друку (print-версія)
e. всі значення в одиницях відносної довжини (%, em/rem, vw, vh і т.п.)
f. кольори у hex
g. адаптивні зображення
h. наявність структурованих даних (або мікроформати або схема JSON-LD)
i. проходить перевірку без помилок HTML та CSS-валідаторами
j. виглядає однаково для основних браузерів (Chrome, Safari, Edge, Firefox)
k. без вбудованих (inline) стилів

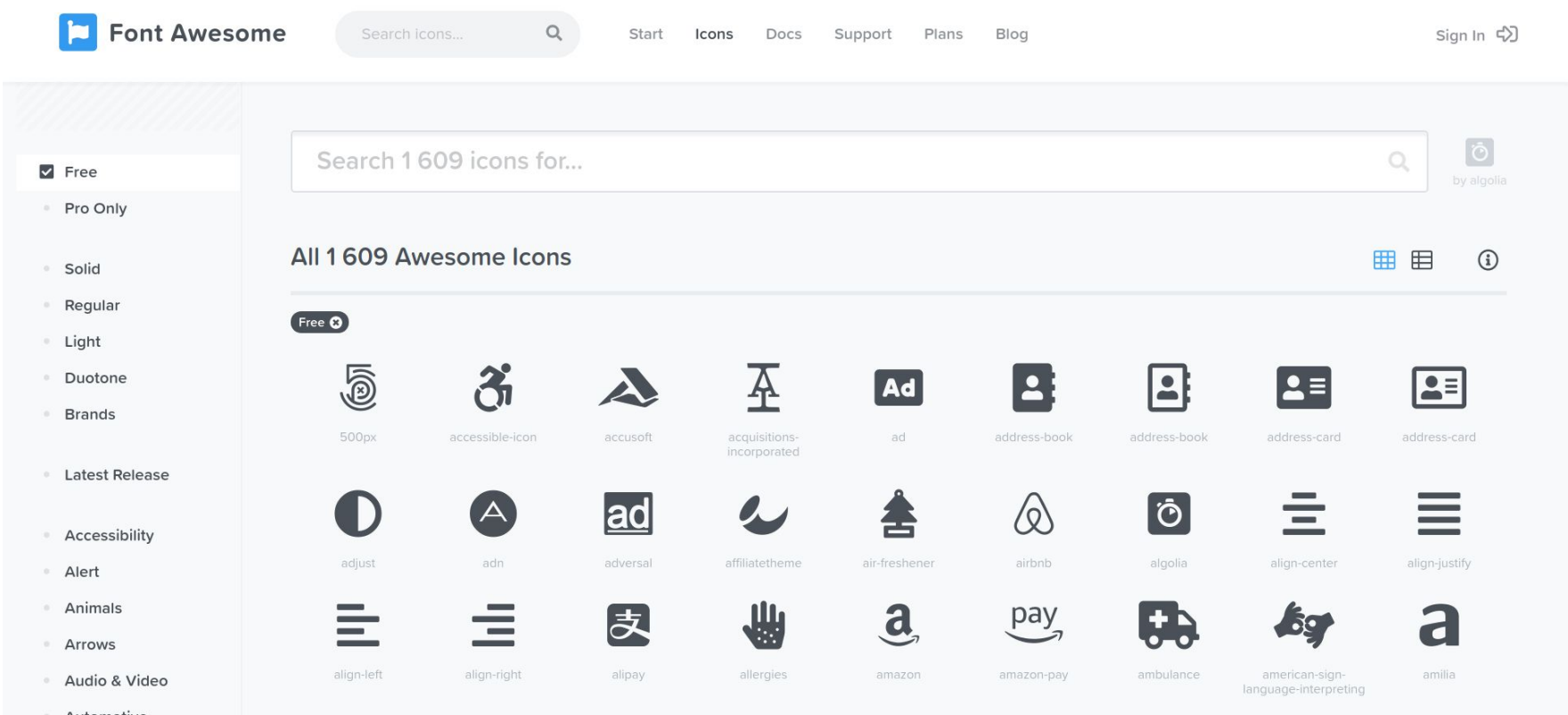
Завдання #3.10

Для обраного дизайну у завданні #3.8 зверстайте сторінку списку записів/постів з підтримкою пагінації (*list*) вашого майбутнього блогу згідно обраного дизайну з наступними вимогами:

a. без використання фреймворку
b. чутлива (responsive) верстка, підтримка різних типів пристроїв та розмірів екрану
c. верстка Mobile First
d. підтримка версії для друку (print-версія)
e. всі значення в одиницях відносної довжини (%, em/rem, vw, vh і т.п.)
f. кольори у hex
g. адаптивні зображення
h. наявність структурованих даних (або мікроформати або схема JSON-LD)
i. проходить перевірку без помилок HTML та CSS-валідаторами
j. виглядає однаково для основних браузерів (Chrome, Safari, Edge, Firefox)
k. без вбудованих (inline) стилів

Бібліотека іконок Font Awesome

https://fontawesome.com/icons



CSS Layout cookbook

Книга рецептів CSS об'єднує приклади більшості найбільш поширених патернів, які можу зустрітися вам при розробці веб-сайтів.

https://developer.mozilla.org/en-US/docs/Web/CSS/Layout_cookbook

Bootstrap

Bootstrap (також відомий як Twitter Bootstrap) - вільний набір інструментів (фреймворк) для створення сайтів і веб-застосунків. Включає в себе HTML- і CSS-шаблони оформлення для типографіки, веб-форм, кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсу, включаючи JavaScript-розширення.

https://getbootstrap.com/

Tailwind CSS

Tailwind CSS - це utility-first CSS-фреймворк для швидкого створення користувацьких інтерфейсів.

Utility-first означає, що, на відміну від Bootstrap, Tailwind не надає нам автоматично попередньо встановлені компоненти і стилі. Він дає нам службові класи, які допомагають стилізувати компонент певним чином, і дозволяє створювати власні класи, використовуючи ці службові класи.

https://tailwindcss.com/