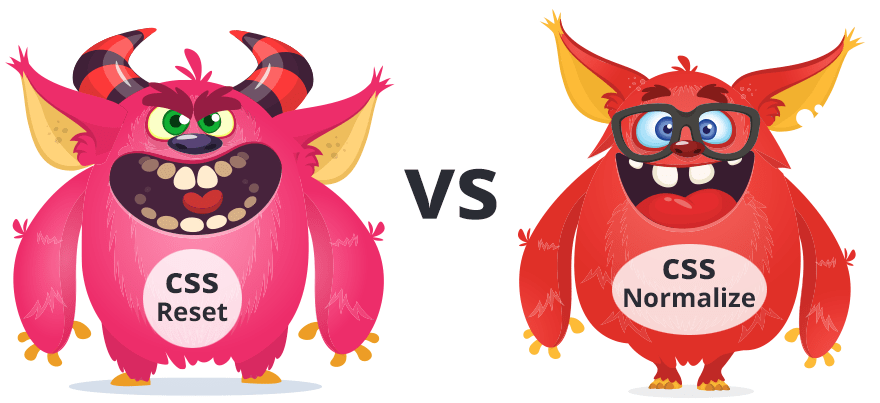
**Лекція №7**

**Нормалізація vs Скидання стилів**

Нормалізація та скидання стилів - це два підходи, які ми використовуємо для стандартизації та контролю відображення веб-сторінок у різних браузерах.

**Нормалізація стилів** полягає у використанні спеціалізованого набору CSS правил, які уніфікують відображення основних HTML елементів у всіх браузерах. Мета нормалізації - забезпечити послідовне відображення елементів, не видаляючи корисні за замовчуванням стилі. Це дозволяє нам побудувати стиль сайту на солідній та уніфікованій основі.

**Скидання стилів (CSS Reset)**, навпаки, видаляє всі вбудовані стилі браузера для HTML елементів, "скидаючи" їх до базового стилю. Це дає нам чисте полотно, на якому ми можемо почати власне стилізування від нуля, уникнення несподіваних розбіжностей у відображенні між різними браузерами.



Обидва ці підходи є важливими у веб-розробці, оскільки допомагають нам забезпечити більш однорідний і передбачуваний досвід користувачів на різних пристроях та у різних браузерах. Вибір між нормалізацією та скиданням стилів залежить від конкретних вимог та цілей проєкту.

**normalize.css**

**Normalize.css** - це маленька CSS бібліотека, яку ми використовуємо для забезпечення більшої узгодженості у відображенні веб-сторінок у різних браузерах. Вона служить альтернативою традиційному CSS Reset. Замість того, щоб повністю скидати стилі, Normalize.css вирівнює стилі для широкого спектра HTML елементів, зберігаючи корисні за замовчуванням стилі та виправляючи помилки, які часто виникають у різних браузерах.

Ми використовуємо Normalize.css, оскільки він надає більш природний стартовий пункт для стилізації веб-сайтів. Він допомагає уникнути багатьох загальних проблем, пов'язаних з кросбраузерністю, забезпечуючи, що всі елементи відображаються консистентно у різних середовищах. Це робить процес розробки більш ефективним, знижуючи необхідність виправлення індивідуальних браузерних особливостей.

<https://www.npmjs.com/package/normalize.css>

**reset.css**

**Reset.css** - це інструмент, який ми використовуємо у веб-дизайні для нівелювання вбудованих стилів браузера для HTML елементів. Це CSS файл, який "скидає" усі стандартні маржини, паддинги, шрифти та інші стилі, які браузери за замовчуванням застосовують до HTML елементів.

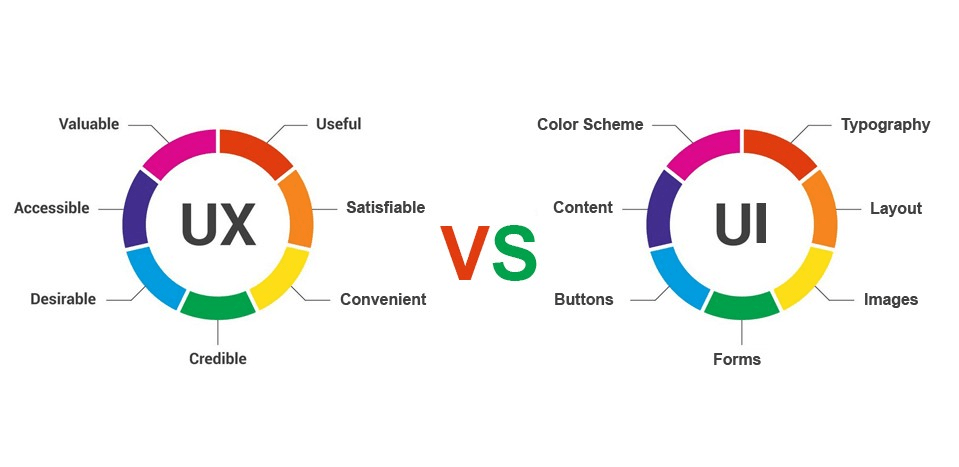
Метою використання reset.css є створення чистої та однорідної основи для подальшої стилізації веб-сайту. Він дозволяє нам почати роботу зі сторінкою, на якій усі елементи мають однаковий вигляд в різних браузерах, що спрощує процес дизайну та розробки. Це дуже корисно, коли ми хочемо мати повний контроль над візуальними стилями веб-сторінки, оскільки ми не змушені боротися з несподіваними або небажаними стилями, які накладають різні браузери.

<https://www.npmjs.com/package/reset-css>

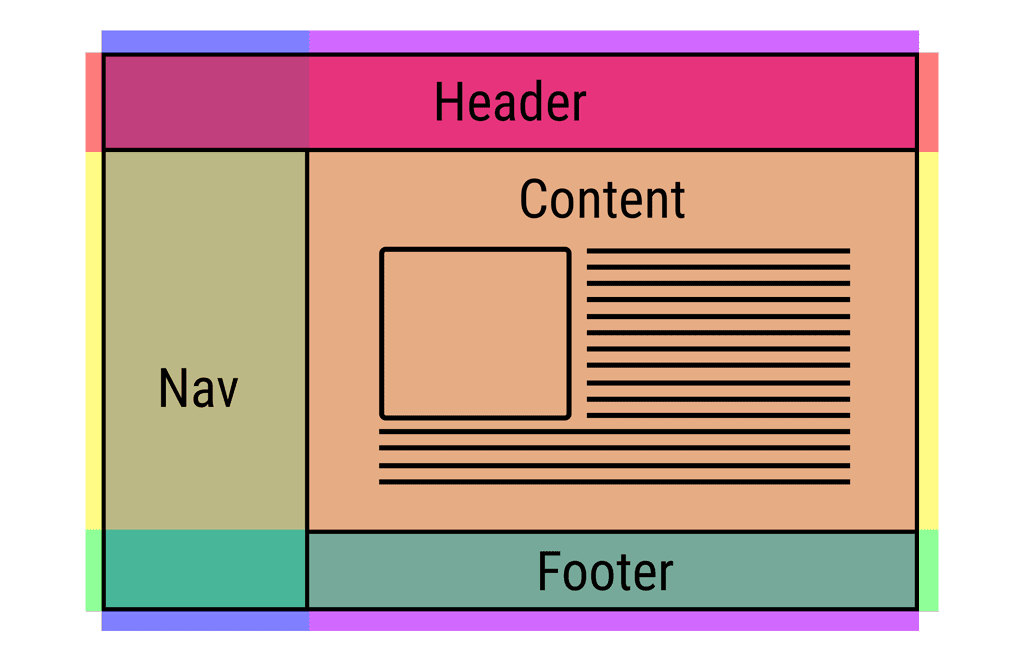
## UI/UX vs Layouts

**UI/UX** відноситься до інтерфейсу користувача (UI) та досвіду користувача (UX) у дизайні веб-сайтів та програмного забезпечення. **Layouts** у контексті веб-дизайну - це схеми або шаблони, які ми використовуємо для організації контенту на веб-сторінках. Вони включають розміщення різних елементів, таких як заголовки, текст, зображення та форми, визначаючи, як буде виглядати та функціонувати веб-сайт. Тобто:

* **UI** (User Interface) - це все, що користувач бачить і з чим взаємодіє безпосередньо на веб-сторінці або в додатку. Це можуть бути кнопки, текстові поля, зображення, слайдери тощо. UI відповідає за візуальний аспект продукту.
* **UX** (User Experience) - це загальний досвід користувача при використанні продукту. UX включає в себе не лише взаємодію з UI, але й такі аспекти, як швидкість завантаження сторінки, інтуїтивність навігації, задоволення від використання продукту та інше.



* Layout - це спосіб організації елементів UI на сторінці або в додатку. Layout визначає, де і як розташовані елементи UI, а також як вони взаємодіють між собою. Layout - це частина UI, але не весь UI.



Отже, UI, UX та Layout взаємопов’язані, але вони фокусуються на різних аспектах дизайну продукту. UI та Layout стосуються візуального представлення та взаємодії з продуктом, тоді як UX охоплює загальний досвід користувача.

## Основні теги для верстки: <div>, <span>

**Div** та **Span** є двома фундаментальними HTML тегами, які ми часто використовуємо у веб-верстці.

**Div** (або тег <div>) - це блоковий елемент, який використовується для створення структури веб-сторінок. Ми застосовуємо його для групування різних елементів та створення "контейнерів" на веб-сторінці. Тег **div** є надзвичайно гнучким, дозволяючи нам застосовувати CSS стилі для форматування розділів сторінки, таких як заголовки, навігаційні панелі, основний контент та футери.

**Span** - це інлайновий(вбудований або рядковий) елемент, представлений тегом <span>. Ми використовуємо **span** для стилізації або відмінності частин тексту в межах інших елементів, не створюючи при цьому нових блоків. Наприклад, ми можемо використати **span** для зміни кольору, шрифту або інших стилістичних характеристик окремих слів або фраз у межах абзацу, не порушуючи при цьому загального потоку документа.

Обидва ці теги є незамінними у веб-розробці, оскільки вони дозволяють нам відокремлювати, групувати та стилізувати контент на сторінці, що є ключовим для створення чітко структурованих та візуально привабливих веб-сайтів.

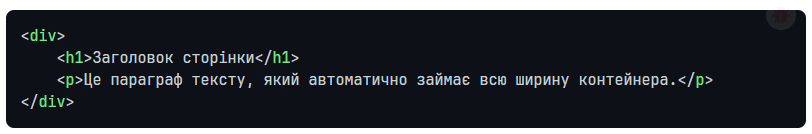
**Блокові та Інлайнові елементи**

**Блокові елементи**

**Блокові елементи** в HTML - це ті елементи, які автоматично починаються з нового рядка на веб-сторінці та розтягуються на всю доступну ширину контейнера. Ми використовуємо блокові елементи для створення основних структурних частин веб-сторінки, таких як заголовки, параграфи та розділи.

Блокові елементи є фундаментальними у веб-дизайні, оскільки вони допомагають нам організовувати контент на сторінці в чітку і логічну структуру. Вони важливі для визначення макету веб-сайту, дозволяючи нам розділити контент на зрозумілі та візуально відокремлені блоки.

**Приклад коду:**

У цьому прикладі:

* <div> - блоковий контейнер, який може вміщувати інші елементи.
* <h1> - блоковий елемент для заголовка, який автоматично починається з нового рядка.
* <p> - блоковий елемент для параграфа, який також починається з нового рядка і розтягується на всю доступну ширину.

Кожен з цих елементів відіграє свою роль у створенні структурованого і зрозумілого макету веб-сторінки.

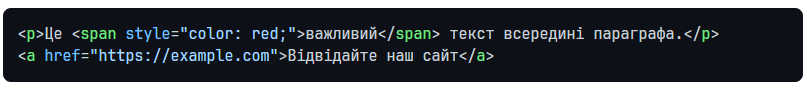
**Інлайнові елементи**

**Інлайнові елементи** в HTML - це ті елементи, які відображаються у межах поточного рядка тексту, не викликаючи автоматичного переносу на новий рядок. Ми використовуємо інлайнові елементи для внесення змін в окремі частини тексту, такі як виділення, зміна кольору, вставка посилань, без порушення загальної структури текстового потоку.

Інлайнові елементи ідеально підходять для розміщення в тексті коротких фрагментів, що вимагають особливого форматування або функціональності, таких як гіперпосилання, сильний акцент або текстові відзначення. Вони допомагають нам робити текст більш інтерактивним і візуально привабливим, не змінюючи його основної структури.

Прикладами інлайнових елементів є <span>, <a>, <strong>, і <em>.

**Приклад коду:**

У цьому прикладі:

* <span style="color: red;"> - інлайновий елемент, що використовується для зміни кольору частини тексту. Цей елемент вписується у потік параграфа (<p>) без створення нового рядка.

<a href="https://example.com" - інший інлайновий елемент, який використовується для створення гіперпосилання. Він також інтегрується у потік тексту без зайвих перерв.

Ці інлайнові елементи дозволяють нам ефективно працювати з текстом, забезпечуючи гнучкість у стилізації та інтерактивності вмісту на веб-сторінці.

**Відмінності між блоковими та інлайновими елементами**

Блокові та інлайнові елементи відрізняються в HTML та CSS за декількома ключовими характеристиками:

**Початок нового рядка:**

* **Блокові елементи** автоматично починаються з нового рядка. Кожен блоковий елемент відображається на новому рядку і займає всю доступну ширину контейнера:
* **Інлайнові елементи** не починають новий рядок. Вони вписуються в потік змісту всередині інших елементів і займають лише стільки місця, скільки необхідно для вмісту.

**Зміна розмірів:**

* **Блокові елементи** можуть мати змінені ширину та висоту.
* **Інлайнові елементи** не підтримують зміну ширини та висоти, оскільки вони займають лише стільки місця, скільки потрібно для їх вмісту.
* **Вертикальне вирівнювання:Блокові елементи** можуть мати маржини та паддинги з усіх боків, включаючи вертикальні.
* **Інлайнові елементи** підтримують горизонтальні маржини та паддинги, але вертикальні маржини та паддинги часто не впливають на інші елементи так, як це роблять блокові елементи.

**Приклади:**

* **Блокові елементи:** <div>, <p>, <h1>...<h6>, <ul>, <ol>, <li>, <section>, <article>, <header>, <footer>.
* **Інлайнові елементи:** <span>, <a>, <img>, <strong>, <em>, <br>, <input>.

**Використання:**

* **Блокові елементи** використовуються для створення основних структурних частин веб-сторінки.
* **Інлайнові елементи** часто застосовуються для форматування тексту або вмісту всередині блокових елементів.

Розуміння цих відмінностей є ключовим для ефективного структурування та стилізації веб-сторінок.

**Властивість display та Формування блокової моделі**

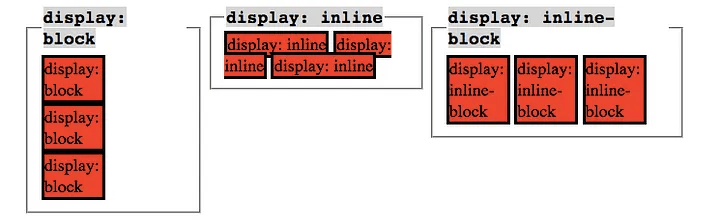
**Властивість display**

**Властивість** display у CSS - це механізм, який ми використовуємо для контролю над способом відображення елементів на веб-сторінці. Ця властивість визначає, як елемент має бути показаний у документі, та дозволяє нам змінювати поведінку елементів, перетворюючи їх з блокових на інлайнові, або навпаки, а також приховувати їх або змінювати на інші типи відображення.

Ми використовуємо display для досягнення різних цілей у веб-дизайні:

1. **Зміна типу елемента:** Наприклад, перетворення інлайнового елемента (display: inline;) в блоковий (display: block;) або навпаки.
2. **Створення гнучких макетів:** За допомогою display: flex; або display: grid; ми можемо створювати складні, адаптивні макети, які реагують на зміни розміру екрану.
3. **Приховування елементів:** З display: none; елемент зникає з веб-сторінки та не займає простору, на відміну від visibility: hidden;, де елемент залишається видимим, але стає прозорим.
4. **Створення інлайнових блоків:** display: inline-block; дозволяє елементам вирівнюватися як інлайнові, але мати властивості блокових елементів.

Розуміння та вміння застосовувати властивість display є важливим для ефективної верстки та дизайну веб-сторінок, оскільки це дає нам більший контроль над тим, як елементи розміщуються та взаємодіють між собою.



**Формування блокової моделі**

Формування блокової моделі (Box Model) в CSS - це фундаментальний принцип, який ми використовуємо для проектування та розташування елементів на веб-сторінці. Вона описує, як властивості, такі як ширина, висота, відступи (margins), рамки (borders) та внутрішні поля (padding), взаємодіють та впливають на загальні розміри та розташування HTML елементів.

**Основні компоненти блокової моделі**

1. **Контент (Content):** Центральна частина, яка включає фактичний вміст елемента, наприклад, текст або зображення. Ширина та висота контенту можуть бути визначені за допомогою властивостей width та height.
2. **Внутрішнє поле (Padding):** Простір між контентом та рамкою. Внутрішнє поле впливає на розмір елемента, але є прозорим.
3. **Рамка (Border):** Обрамлення навколо внутрішнього поля та контенту. Розмір та стиль рамки можна налаштувати.
4. **Відступи (Margin):** Простір між рамкою елемента та іншими елементами. Відступи не впливають на розмір самого елемента, але визначають відстань між елементами.

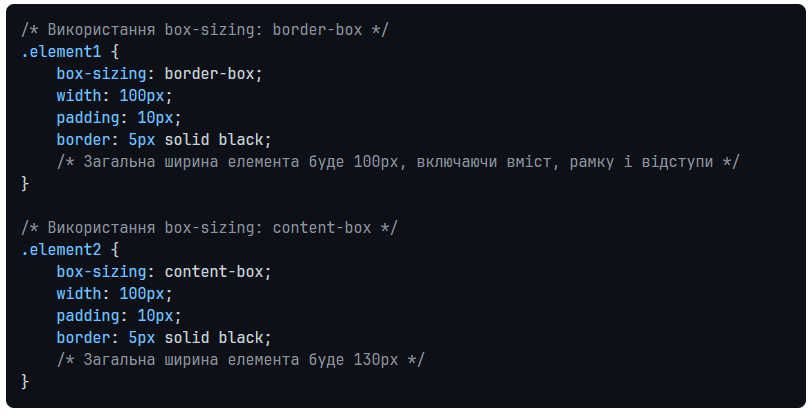
Блокова модель є ключовою для розуміння того, як розміщувати та вирівнювати елементи на сторінці, особливо коли створюється макет із різними блоками контенту та інтерактивними компонентами. Використання цієї моделі дозволяє нам створювати акуратно організовані, візуально привабливі та зручні для користувача веб-інтерфейси.

**box-sizing**

**Box-sizing** є CSS властивістю, яка визначає, як розмір елемента має бути обчислений у відношенні до його відступів (padding), рамок (border) та внутрішнього вмісту (content). Ця властивість приймає два основних значення: border-box та content-box, які змінюють спосіб обчислення загальних розмірів елемента.

1. box-sizing: border-box; При цьому налаштуванні загальна ширина та висота елемента включають відступи та рамки. Це означає, що якщо ми встановлюємо елементу ширину 100px і додаємо рамку шириною 10px та відступ 5px, загальна ширина елемента залишиться 100px, а вміст відповідно зменшиться.
2. box-sizing: content-box; Це значення є стандартним і означає, що ширина та висота елемента відносяться тільки до вмісту, не включаючи рамки та відступи. Так, якщо ми встановлюємо ширину елементу 100px, додаємо рамку 10px та відступ 5px, загальна ширина елемента становитиме 130px (100px вміст + 10px рамка з кожного боку + 5px відступ з кожного боку).

**Приклад коду:**

/\* Використання box-sizing: border-box \*/

.element1 {

box-sizing: border-box;

width: 100px;

padding: 10px;

border: 5px solid black;

/\* Загальна ширина елемента буде 100px, включаючи вміст, рамку і відступи \*/

}

/\* Використання box-sizing: content-box \*/

.element2 {

box-sizing: content-box;

width: 100px;

padding: 10px;

border: 5px solid black;

/\* Загальна ширина елемента буде 130px \*/

}

У першому випадку .element1 з border-box загальна ширина елемента залишиться 100px, незалежно від відступів та рамок. У другому випадку .element2 з content-box загальна ширина становитиме 130px, оскільки до 100px ширини вмісту додаються рамки та відступи. Вибір між цими двома властивостями залежить від потреб макету та переваг розробника.

**Обтікання елементів (Float)**

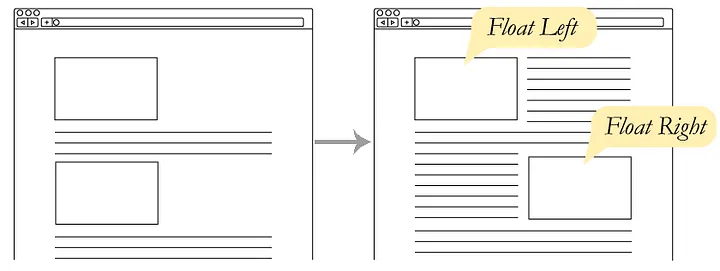
**Обтікання елементів (Float)** у CSS - це техніка, яку ми використовуємо для розміщення елементів у межах контейнера так, що вони плавають або обтікають один одного. Зазвичай використовується для зображень, тексту та інших елементів, що дозволяє створити макети, де текст обтікає зображення або інші елементи візуально впорядковуються у певний спосіб.

Ми застосовуємо float, щоб відокремити елемент від звичайного потоку документа та дозволити іншим елементам обтікати його зліва або справа. Ця властивість корисна для створення компонувань, де елементи, як-от текст та зображення, взаємодіють тісніше, ніж це можливо зі стандартним блоковим чи інлайновим відображенням.

**Основні способи використання float**

1. float: left; - елемент обтікається зліва, дозволяючи іншому контенту обтікати його з правого боку.
2. float: right; - елемент обтікається справа, дозволяючи іншому контенту обтікати його з лівого боку.
3. float: none; - стандартне значення, елемент не обтікається та слідує звичайному потоку документа.

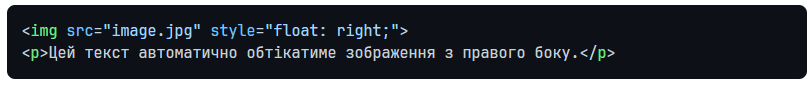
Обтікання використовується, коли нам потрібно, щоб контент ефективно інтегрувався з іншими елементами сторінки, створюючи гармонійний і збалансований візуальний макет.



**Приклади float**

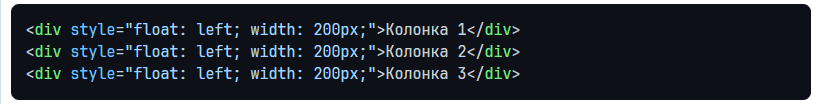
Декілька прикладів, як ми використовуємо float у CSS для різних завдань:

**Обтікання текстом зображення:** Якщо ми хочемо, щоб текст обтікав зображення зліва або справа, ми можемо застосувати float до зображення. Наприклад:

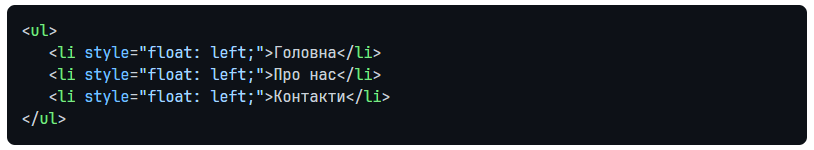


Створення багатоколонкового макету:

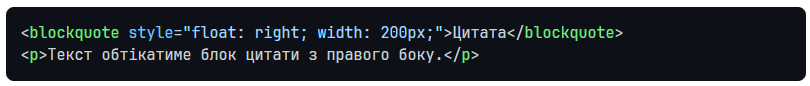
Ми можемо використати float для створення горизонтально розташованих блоків, як у макеті з декількома колонками:

Створення навігаційного меню:

Float можна використовувати для розміщення елементів навігаційного меню горизонтально:

Обтікання текстом інших елементів:

Ми можемо використовувати float, щоб текст обтікав не тільки зображення, а й інші елементи, наприклад, цитати або бічні панелі:

У всіх цих прикладах float допомагає нам створювати макети, де елементи взаємодіють між собою гнучко, дозволяючи тексту та іншому контенту обтікати плаваючі об'єкти, чим забезпечується більш динамічний та інтерактивний веб-дизайн.

**Заборона обтікання & Властивість overflow**

**Заборона обтікання**

Заборона обтікання в CSS використовується для скасування ефекту float, що застосований до попередніх елементів на веб-сторінці. Це важливо для відновлення нормального потоку документа після використання обтікання. Це зазвичай досягається за допомогою властивості clear.

Властивість clear використовується для вказівки, з якого боку елементу не повинно бути обтікання (float). Це може бути ліва, права сторона або обидві. Ось основні варіанти використання clear:

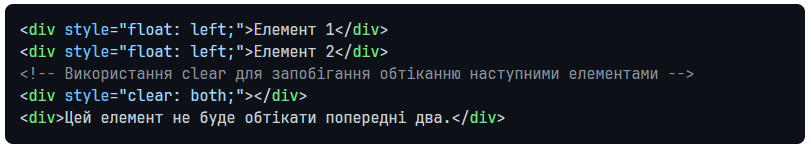
1. **clear: left;** - забороняє обтікання з лівого боку елемента.
2. **clear: right;** - забороняє обтікання з правого боку елемента.
3. **clear: both;** - забороняє обтікання з обох боків елемента.

Ця властивість часто використовується після ряду плаваючих елементів для забезпечення того, що наступний контент не буде обтікати ці елементи, а відобразиться під ними, як це передбачено нормальним потоком документа.

Error

**clear**

Приклад використання clear:

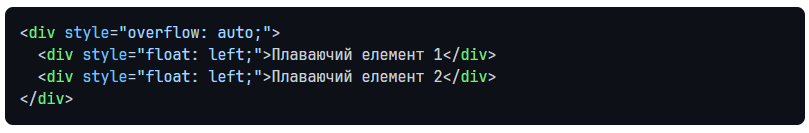
У цьому прикладі, третій div з clear: both; запобігає обтіканню наступного div попередніми двома плаваючими елементами, забезпечуючи чистий перехід до наступного блоку контенту.

Error

**overflow**

Властивість overflow в CSS іноді може використовуватися як техніка для вирішення проблем, пов'язаних з обтіканням (float), хоча її основна мета - контролювати поведінку контенту, який виходить за рамки свого контейнера.

**Приклад використання overflow:**

Тут, застосування overflow: auto; до батьківського div дозволяє йому повністю обгортати обидва плаваючі елементи, незалежно від їхніх розмірів. Це допомагає уникнути непередбачених ефектів у макеті, таких як перекриття чи розмірні неузгодженості, що можуть виникнути через плаваючі елементи.

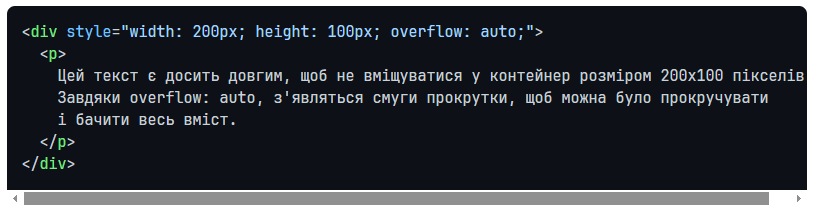
**Властивість overflow**

Властивість **overflow** в CSS визначає, як ми обробляємо вміст елемента, який виходить за межі визначених розмірів його контейнера. Ця властивість допомагає нам контролювати поведінку вмісту, який переповнює свій блок, дозволяючи вирішити, чи повинен цей вміст бути обрізаний, прокручуваним, або чи повинен він залишитися видимим.

Ми використовуємо overflow для вирішення різних завдань:

1. **Для запобігання виходу вмісту за межі контейнера:** Якщо ми хочемо, щоб вміст, який не вміщується в блок, не був видимим, ми можемо використати overflow: hidden;.
2. **Для створення прокручуваного контейнера:** Якщо вміст повинен бути доступним через вертикальну або горизонтальну прокрутку, ми встановлюємо overflow: scroll; або overflow: auto;, де останнє автоматично відображає смуги прокрутки, коли це необхідно.
3. **Для візуальних ефектів та анімації:** Наприклад, можна створити ефект "вспливання" за допомогою обрізання вмісту та його анімації в контейнері з overflow: hidden;.

**Приклад коду з описом:**

У цьому прикладі, контейнер з фіксованою шириною та висотою містить абзац тексту, який є занадто довгим для відображення без прокрутки. Властивість overflow: auto; забезпечує появу смуг прокрутки в контейнері, якщо весь вміст не може бути показаний одночасно, дозволяючи користувачу прокручувати його для перегляду.

**Позиціонування блоків**

**Позиціонування блоків** в CSS – це спосіб, яким ми визначаємо місце елементів на сторінці. Використовуючи властивості позиціонування, ми можемо точно розмістити елементи, незалежно від звичайного потоку документа.

Це використовується для різних цілей:

* **Створення надійних макетів:** Встановлення фіксованих позицій для навігаційних панелей чи кнопок.
* **Налаштування взаємодії елементів:** Розміщення вспливаючих вікон або модальних діалогів.
* **Складні візуальні ефекти:** Наприклад, налаштування різних елементів на сторінці для створення паралакс-скролінгу.

**Типи позиціонування**

Є кілька основних типів позиціонування в CSS:

* **Static:** Стандартне позиціонування, де елементи розташовуються в звичайному потоці документа.
* **Relative:** Елемент зміщується відносно його нормального положення.
* **Absolute:** Елемент розташовується відносно найближчого позиціонованого предка (не **static**).
* **Fixed:** Елемент розташовується відносно вікна переглядача і не рухається під час прокрутки.
* **Sticky:** Комбінація відносного та фіксованого позиціонування.

**Вказівки позиції елемента**

**Left, top, right, і bottom** це CSS властивості, які ми використовуємо для вказівки позиції елемента у відношенні до його найближчих позиціонованих предків. Вони працюють тільки для елементів, що мають позиціонування, відмінне від static, тобто relative, absolute, fixed або sticky.

* left **і** right контролюють горизонтальне позиціонування елемента.
* top **і** bottom контролюють вертикальне позиціонування.

Ці властивості дають нам можливість точно розташовувати елементи на сторінці, як наприклад, при створенні спливаючих вікон або під час стилізації форм на сторінці.

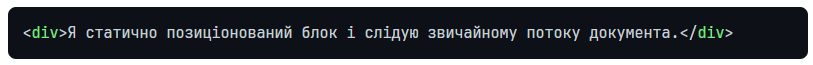
**z-index** - це CSS властивість, яка визначає стековий порядок елементів, тобто як елементи накладаються один на одного. Велике значення z-index означає, що елемент буде розташований вище в стековій послідовності, тобто "вище" на сторінці. Ми використовуємо z-index для управління перекриттям елементів, як наприклад, коли потрібно забезпечити, щоб модальне вікно або спливаюче меню залишались над іншими елементами на сторінці.

Всі ці властивості є необхідними інструментами в нашому наборі для розробки веб-інтерфейсів, що вимагають складного позиціонування та забезпечення правильного візуального шарування елементів.

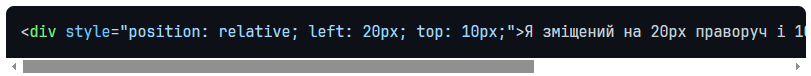
**Приклади застосування:**

Приклади застосування різних значень властивості position в CSS:

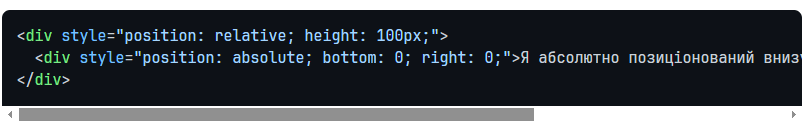
**Static** (статичне позиціонування) – це значення за замовчуванням, яке не потребує явного визначення, оскільки елементи автоматично розташовуються в звичайному потоці документа.



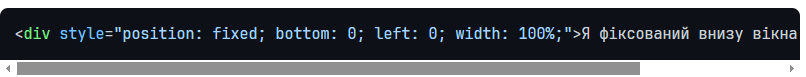
**Relative** (відносне позиціонування) – елемент зміщується відносно його початкової позиції у нормальному потоці документа, не впливаючи на розміщення інших елементів.



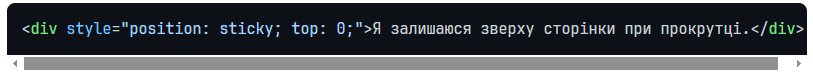
**Absolute** (абсолютне позиціонування) – елемент видаляється з нормального потоку документа і розташовується відносно його найближчого позиціонованого предка (не static).



**Fixed** (фіксоване позиціонування) – елемент фіксується на одному місці вікна переглядача і не рухається під час прокрутки сторінки.



**Sticky** (липке позиціонування) – елемент є відносно позиціонованим до тих пір, поки не зустрічається визначений поріг у вигляді прокрутки сторінки, після чого він поводиться як фіксований.

Кожне з цих значень властивості position використовується для досягнення певної поведінки в макеті веб-сторінки і має своє власне унікальне застосування залежно від потреб розробника і дизайну сайту.

Error

**Fixed vs Sticky**

**Sticky (липке позиціонування)** - це гібрид між відносним та фіксованим позиціонуванням. Елемент з position: sticky; розташовується як відносно позиціонований до тих пір, поки він не досягає встановленого порогу (зазначеного за допомогою top, bottom, left чи right), у рамках свого батьківського контейнера. Після досягнення цього порогу при прокрутці сторінки, елемент стає фіксованим до тих пір, поки його батьківський блок залишається у видимій області вікна переглядача.

Відмінність від fixed полягає в тому, що sticky елемент "прилипає" лише при досягненні певного положення при прокрутці, а fixed елемент завжди залишається на одному місці незалежно від прокрутки сторінки.

**Приклад коду:**

<!DOCTYPE html>

<html>

<head>

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

.fixed-header {

position: fixed;

top: 0;

left: 0;

width: 100%;

background-color: #333;

color: white;

text-align: center;

padding: 10px;

z-index: 1000;

}

.content {

padding: 50px;

margin-top: 40px; /\* Відступ для компенсації фіксованого заголовка \*/

background-color: #f4f4f4;

}

.sticky-div {

position: -webkit-sticky;

position: sticky;

top: 40px; /\* Позиція, де прилиплий елемент стане фіксованим \*/

background-color: yellow;

padding: 10px;

font-size: 20px;

margin-top: 20px;

border: 1px solid black;

}

</style>

</head>

<body>

<div class="fixed-header">Fixed Header (Завжди вгорі)</div>

<div class="content">

<p>Деякий контент перед прилиплим блоком...</p>

<div class="sticky-div">Sticky Div (Прилипає при прокрутці)</div>

<p>Додатковий контент, що демонструє поведінку прилиплого блоку...</p>

<div style="height: 1000px;"></div>

<p>Кінець контенту.</p>

</div>

<div style="height: 1000px;"></div>

</body>

</html>



**Пояснення:**

* **Fixed Header**: Цей заголовок використовує position: fixed і завжди залишається на верхній частині вікна браузера, незалежно від того, де ви знаходитесь на сторінці.
* **Sticky Div**: Цей блок використовує position: sticky. Він спочатку поводиться як звичайний блок. Але коли ви прокручуєте сторінку вниз, і він досягає верху вікна браузера (з урахуванням зазначеного top: 40px), він стає "прилиплим" і залишається на видимій частині екрану, доки його батьківський елемент (div.content) перебуває на екрані. Коли ви прокручуєте сторінку далі, після того як весь контент div.content зникає з видимої частини екрану, прилиплий блок знову стає звичайним і зникає з виду.

Цей приклад повинен ясно демонструвати різницю: фіксований заголовок завжди залишається нагорі, незалежно від прокрутки, тоді як прилиплий блок з'являється та зникає в залежності від положення прокрутки сторінки.