МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
“Ужгородський національний університет”
Математичний факультет
Кафедра системного аналізу і теорії оптимізації

ОСНОВИ CSS

Частина 1

Методичні вказівки для самостійної роботи студентів математичного факультету з дисципліни «Вступ до web-програмування»

Ужгород – 2018

Брила А.Ю., Глебена М.І., Ломага М.М., Млавець Ю.Ю. Методичні вказівки для самостійної роботи студентів математичного факультету з дисципліни «Вступ до web-програмування». Ч. 1. Основи CSS.– Ужгород, 2018 . – 54с.

Методичні вказівки є комплексом рекомендацій навчально-методичного характеру з питань організації самостійної роботи студентів та виконання індивідуальних практичних завдань з дисципліни «Вступ до web-програмування». Призначений для студентів спеціальностей 113 Прикладна математика та 124 Системний аналіз.

Рекомендовано до друку Вченою радою математичного факультету

ДВНЗ "Ужгородський національний університет"

від 14 червня 2018 року, протокол № 11.

Вступ

Web-дизайн є основою створення інтерфейсів користувача широкого спектру Web орієнтованих додатків: Web-порталів, інформаційних, довідкових, пошукових, корпоративних, освітніх та інших типів сайтів та систем. Крім того, велика кількість програмного забезпечення, яке функціонує із використанням так званого віконного інтерфейсу користувача, поступово переводиться у Web орієнтований спосіб взаємодії із користувачем для підвищення ефективності розробки та супроводження існуючого програмного забезпечення.

Web-дизайн включає такі важливі складові, як розмітка Web документу, його зовнішнє оформлення, інтерактивність елементів управління сторінки, а також зручність експлуатації Web-сайтів в цілому.

Розглядаючи технічний аспект Web-дизайну, слід відмітити, перш за все, мови розмітки документів – XHTML (eXtensible HyperText Markup Language) та XML (eXtensible Markup Language), які дозволяють вирішити задачу структуризації інформації на Web-сторінках у чіткий формалізований спосіб.

Наступним технічним аспектом є мова стильового оформлення документів CSS (Cascading Style Sheets), яка призначена для надання єдиного візуального стилю усім сторінкам Web-сайту або Web-додатка. Саме мова CSS призначена для винесення функцій форматування текстових, табличних, графічних та інших видів даних з HTML сторінки, інакше кажучи, відокремлення даних від опису їх виведення.

Крім технічних задач, які постають перед розробниками, важливими є також питання естетичного сприйняття інформації, наприклад, композиції елементів, кольорової гами, єдності та контрасту елементів тощо, а також питання зручності експлуатації (Usability).

Якісний сайт стає важливим, а у деяких галузях – єдиним, засобом досягнення економічних, політичних, соціальних, рекламних та інших цілей.

Якісний сайт відрізняється від інших сайтів у глобальній мережі (яких більшість) такими рисами: висока якість інформаційного наповнення й грамотність його подачі; оригінальність і естетична привабливість зовнішнього вигляду сторінок; доступність змісту сайту для максимально широкого кола користувачів поза залежністю від застосовуваних ними типів пристроїв і версій браузерів, а також від обмежень за станом здоров’я; ергономічність елементів користувальницького інтерфейсу сайту, що забезпечує високу ефективність, але в той же час легкість і невимушеність взаємодії відвідувача з веб-ресурсом; надійність і безпека використовуваних технологічних рішень, чітка погодженість роботи всіх компонентів; бездоганне пророблення всіх деталей і нюансів.

Але для створення якісного сайту, тобто відповідності цим рисам, потрібна плідна робота висококваліфікованих спеціалістів із різних Web-технологій, які повинні розуміти не лише вузьку галузь знань але і добре представляти весь спектр Web-технологій.

Необхідним елементом успішного засвоєння навчального матеріалу дисципліни «Вступ до web-програмування» є самостійна робота студентів з технічною літературою, та сучасними програмними засобами розробки програм.

Завданням даних методичних вказівок є допомога в набутті студентами практичних навичок створення сучасних Web-орієнтованих інтерфейсів користувача з використанням загальноприйнятих стандартів, сучасного програмного інструментарію, технологій, мов розмітки та програмування.

Основи CSS

Cascading Style Sheets (каскадні таблиці стилів) − технологія опису зовнішнього вигляду документа, написаного мовою розмітки. CSS використовується переважно для оформлення HTML- і XHTML-документів, але іноді і для інших XML-структурованих документів (наприклад, в браузері Mozilla для оформлення елементів графічного інтерфейсу, XUL).

CSS використовується авторами веб-сторінок для задання кольорів, шрифтів, розташування і інших аспектів представлення документа. Основною метою розробки CSS було розділення вмісту (написаного на HTML або іншій мові розмітки) і представлення документа (написаного на CSS). Це розділення може збільшити доступність документа, надати велику гнучкість і можливість керування його представленням, а також зменшити складність і повторюваність в структурному вмісті. Крім того, CSS дозволяє представляти один і той же документ в різних стилях або методах виводу в залежності від пристрою виводу.

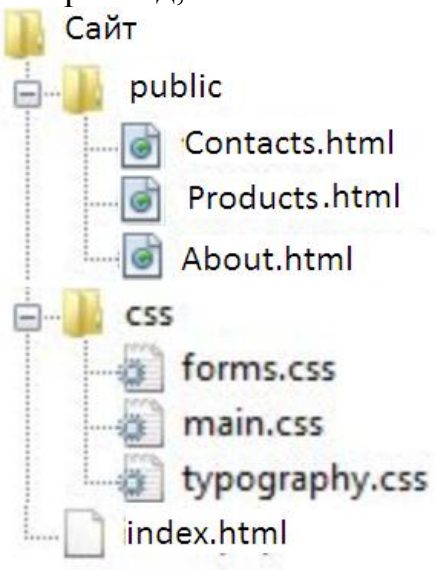
Таблиця стилів складається з набору правил. Стандарт CSS визначає пріоритети, у порядку яких застосовуються правила стилів, якщо для якогось елементу підходять деякі правила одночасно. Це називається "каскадом", в якому для правил розраховуються пріоритети або "ваги", що робить результати передбачуваними.

До появи CSS оформлення веб-сторінок здійснювалося безпосередньо всередині вмісту документа.

```
<p style=”color:red”> Важлива інформація </p>
```

Проте з появою CSS стало можливим принципове розділення вмісту (HTML-розмітка) і представлення (форматування) документа (CSS).

Наприклад,



За рахунок цього нововведення стало можливим легке застосування єдиного стилю оформлення для маси схожих документів, а також швидка зміна цього оформлення.

Переваги CSS розмітки:

- Декілька дизайнів сторінки для різних пристроїв перегляду. Наприклад, на екрані дизайн буде розрахований на велику ширину, під час друку меню не виводитиметься, а на КПК і стільниковому телефоні меню буде розташоване не збоку, а вслід за вмістом.
- Зменшення часу завантаження сторінок сайту за рахунок перенесення правил представлення даних в окремий CSS-файл. В цьому випадку браузер завантажує тільки структуру документа і дані, що зберігаються на сторінці, а представлення цих даних завантажується браузером тільки один раз і кешується.
- Простота подальшої зміни дизайну. Не потрібно правити кожну сторінку, а лише змінити CSS.
- Додаткові можливості оформлення. Наприклад, за допомогою CSS-розмітки можна зробити блок тексту, який решта тексту обтікатиме (наприклад для меню) або зробити так, щоб меню було завжди видно при скролінгу сторінки.

Способи задання стилів

Для додавання стилів на веб-сторінку існує кілька способів, які розрізняються своїми можливостями і призначенням:

- зовнішні (пов’язані) таблиці стилів;
- внутрішні (глобальні) стилі;
- вбудовані стилі.

Кожен із цих способів може застосовуватися як самостійно, так і в поєднанні з іншими на одній сторінці. У цьому випадку необхідно пам'ятати про їх ієрархії. Найбільший пріоритет має вбудований стиль, потім внутрішній, потім зовнішні таблиці стилів.

Зовнішні таблиці стилів

При використанні зв'язаних стилів опис селекторів і їх значень розташовується в окремому файлі, як правило, з розширенням css, а для зв'язування документа з цим файлом застосовується тег `<link>`. Даний тег поміщається в контейнер `<head>`.

Загальний вигляд	**<link rel="stylesheet" type="text/css" href="шлях до файлу">** у HTML5 атрибут type можна не вказувати **<link rel="stylesheet" href="шлях до файлу">**
Приклад	**<head> <link rel="stylesheet" href="css/style.css"> <link rel="stylesheet" href="css/assets.css" media="all"> </head>**

Розглянемо деякі атрибути тегу `<link>`

Атрибут	Опис атрибута
href	Повний або відносний шлях до об'єкту що підключається.

media	Визначає, які стилі застосовувати для конкретного пристрою
rel	Визначає тип взаємозв'язку між поточним html-документом і пов'язаним
type	MIME-тип даних зв'язуваного файлу. У HTML5 за замовчуванням має значення “type/css”, тому при підключенні зовншінх таблиць стилів може бути не вказаним.

Зауважимо, що таким чином можна підключати декілька зовнішніх таблиць стилів. Ці зовнішні таблиці стилів можуть знаходитися і на іншому сайті. Файл із стилем не зберігає ніяких даних, крім синтаксису CSS. У свою чергу і HTML-документ містить тільки посилання на файл зі стилем, тобто таким способом в повній мірі реалізується принцип поділу коду та оформлення сайту. Тому використання пов'язаних стилів є найбільш універсальним і зручним методом додавання стилю на сайт. Адже стилі зберігаються в одному файлі, а в HTML-документах зазначається лише посилання на нього.

Внутрішні (глобальні) стилі

При використанні глобальних стилів властивості CSS описуються у самому документі за допомогою тега-контейнера <style> і розташовуються у заголовку веб-сторінки (в середині тега <head>).

Загальний вигляд	**<head>** **<style type="text/css">** опис стилю згідно синтаксису CSS **</style>** **</head>** у HTML5 атрибут type можна не вказувати
Приклад	**<head>** **<style>** **h2 {** **color: red;** **font-family: "Times New Roman",** **Georgia, Serif;** **line-height: 1.3em;** **}** **...** **</style>** **...** **</head>** **<body>** **...** **</body>**

За своєю гнучкістю і можливостями цей спосіб додавання стилю поступається попередньому, але також дозволяє зберігати стилі в одному місці, в даному випадку прямо на тій же сторінці. Відмітимо, що внутрішні стилі мають

перевагу над зовнішніми. Тобто, якщо у зовнішніх і внутрішніх стилях задано одне і те ж правило для одного і того ж елемента, то буде прийнято до розгляду правило, яке задане у внутрішніх стилях.

Вбудовані стилі

Вбудований стиль є атрибутом елемента на поточній веб-сторінці. Для визначення стилю використовується атрибут `style`, а його значенням виступає набір стильових правил.

Загальний вигляд	**< тег style = "властивість1:значення1; властивість2:значення2;…" >**
Приклад	текст приклад

Даний спосіб задання стилів є найменш гнучким і тому використовується рідко. Зазначимо, що такий спосіб задання має найбільш високий пріоритет у разі використання декількох способів задання стилів на одній сторінці.

Правило @import

Існує ще і інший спосіб імпорту зовнішніх таблиць стилів у файли HTML, який полягає у використані оператора `@import`.

Приклад

```
<style>
 @import url("styles.css");

 ... інші оператори імпорту або стилі CSS...
</style>
```

Іноді ви можете побачити оператори імпорту без дужок, але це означає те ж саме. Необхідно знати ще, що оператор @import повинен завжди бути першим у вкладеній таблиці стилів. Існують невеликі незначні відмінності між використанням `@import` та `link`, але вони дуже незначні, тому можна вибрати будь-який спосіб. Хоча елементи link вважаються на сьогодні кращим способом.

`@import` може бути використаним для підключення шрифтів:

```
@import url(https://fonts.googleapis.com/css?
family=Open+Sans &subset=latin,cyrillic);
```

`@import` також використовують для підлючення (вставки) одних таблиць стилів в інші.

Синтаксис CSS

Таблиця стилів складається з набору правил. Кожне правило, у свою чергу, складається з одного або декількох селекторів і блоку визначень, що відносяться до них. *Селектор* – це деяке ім'я стилю, для якого додаються параметри форматування. Селектори описують ту частину веб-сторінки, до якої необхідно застосувати вказані правила форматування. В якості селектора можна вказати:

- елементи (теги);
- класи;
- ідентифікатори;
- псевдокласи;
- псевдоелементи.

Загальний вигляд	**селектор{** **властивість1:значення1;** **властивість2:значення2;** … **властивістьN:значення;** **}**
Приклад	**body {** **color: red;** **font-weight: normal;** **}** **h1, h2 {** **color: #00ff00;** **}** **p {** **color: rgb(0,0,255);** **}**

Зауважимо, що крапку з комою в кінці можна опустити.

ПРИМІТКИ:

■ CSS не чутливий до регістру, перенесення рядків, пробілів і символів табуляції, тому форма запису залежить від бажання розробника.

Приклад. Нижче описані ідентичні стилі

```
 h1 { color: #a6780a; font-weight: normal; border-bottom: 2px
solid black}
 H1 {
 border-bottom: 2px solid black
 color: #a6780a;
 font-weight: normal;
 }
```

■ Для селектора допускається додавати кожне правило окремо:

```
td {background: olive;}
```

```
td {color: white;}
td {border: 1px solid black;}
```

Однак такий запис не дуже зручний. Доводиться повторювати кілька разів один і той же селектор, та й легко заплутатися в їх кількості. Тому пишіть всі властивості для кожного селектора разом. Компактна форма запису

```
td {
 background: olive;
 color: white;
 border: 1px solid black;
}
```

■ Якщо для селектора спочатку задається властивість з одним значенням, а потім та ж властивість, але вже з іншим значенням, то застосовуватися буде те значення, яке в коді встановлено останнім

```
p {color: green;}
p {color: red;}
```

У даному прикладі для селектора `p` колір тексту спочатку задається зеленим, а потім червоним. Оскільки значення `red` розташоване нижче, то воно в підсумку і буде застосовуватися до тексту.

Насправді такого запису краще взагалі уникати і видаляти повторювані значення. Але подібне може статися неявно, наприклад, у випадку підключення різних стильових файлів, в яких містяться однакові селектори.

У кожної властивості може бути тільки відповідне його функції значення. Наприклад, для `color`, який встановлює колір тексту, в якості значень неприпустимо використовувати числа.

Коментарі

Щоб позначити, що текст є коментарем, застосовують наступну конструкцію `/ * …коментар... * /`

Загальний вигляд	/ * …коментар... * /

Приклад	/*
	**********Мій стиль**********
	Зроблено для ознайомлення
	*/
	body {
	color: red;
	}
	p {
	color: rgb(0,0,255);
	}

Коментарі потрібні, щоб робити пояснення з приводу використання тієї чи іншої стильової властивості, виділяти розділи або писати свої нотатки. Коментарі дозволяють легко згадувати логіку і структуру селекторів, і підвищують розбірливість коду. Разом з тим, додавання тексту збільшує обсяг

документів, що негативно позначається на часі їх завантаження. Тому коментарі зазвичай застосовують в налагоджувальних або навчальних цілях, а при викладанні сайту в мережу їх вилучають. Коментарі можна додавати у будь-яке місце CSS-документа, а також писати текст коментаря у кілька рядків. Вкладені коментарі недопустимі.

Типи даних

У якості значення для властивостей можуть бути вказані величини таких типів даних:

- рядки;
- числа;
- відносні або абсолютні одиниці виміру;
- адреси;
- кольори.

Рядки

Будь-які рядки необхідно брати в подвійні лапки. Якщо всередині рядка потрібно використати лапки чи слеш, то необхідно додати перед ними додатковий слеш (екранувати символ).

Приклад.

```
"Готель \" Турист \" "
```

Числа

Значенням може виступати ціле число, що містить цифри від 0 до 9 і десятковий дріб, в якій ціла і десяткова частина розділяються крапкою.

Приклад.

```
500
3.14
```

Розміри

Для задання розмірів різних елементів, в CSS використовуються *абсолютні* і *відносні* одиниці виміру. Абсолютні одиниці не залежать від пристрою виводу, а відносні одиниці визначають розмір елементу щодо значення іншого розміру.

Відносні одиниці зазвичай використовують для роботи з текстом, або коли треба обчислити співвідношення між елементами.

em	Розмір шрифту поточного елементу
ex	Висота символу x
px	Піксель
%	Відсоток

Одиниця **em** – це змінюване значення, яке залежить від розміру шрифту поточного елементу (розмір встановлюється через стильову властивість font-size). В кожному браузері закладений розмір тексту, який застосовується в тому випадку, коли цей розмір явно не заданий. Тому спочатку 1**em** дорівнює розміру шрифту, заданого в браузері за замовчуванням або розміром шрифту батьківського елементу. Відсотковий запис ідентичний до **em**, в тому сенсі, що значення 1**em** і **100%** рівні.

Одиниця **ex** визначається як висота символу «x» в нижньому регістрі. На **ex** розповсюджуються ті ж правила, що і для **em**, а саме, він прив'язаний до розміру шрифту, заданого в браузері за замовчуванням, або до розміру шрифту батьківського елементу.

Піксель − це точка певного кольору, що відображається монітором або іншим подібним пристроєм, наприклад, смартфоном. Розмір пікселя залежить від розширення пристрою і його розмірів.

Відсотковий запис зазвичай застосовується в тих випадках, коли треба змінити значення щодо батьківського елементу або коли розміри залежать від зовнішніх умов. Так, ширина таблиці 90% значить, що вона буде підлаштовуватися під розміри вікна і змінюватися разом з шириною вікна.

Абсолютні одиниці застосовуються рідше, ніж відносні і зазвичай при роботі з текстом.

in	Дюйм (1 дюйм дорівнює 2,54 см)
cm	Сантиметр
mm	Міліметр
pt	Пункт (1 пункт дорівнює 1/72 дюйма)
pc	Піка (1 піка дорівнює 12 пунктам)

Найпоширенішою одиницею є пункт, який використовується для вказівки розміру шрифту. Це єдина величина з неметричної системи вимірювання, яка має широке використання. І все завдяки текстовим редакторам і видавничим системам.

Адреси

(URI, Uniform Resource Identifiers, уніфікований ідентифікатор ресурсів) застосовуються для вказівки шляху до файлу, наприклад, для установки фонової картинки на сторінці. Для цього застосовується ключове слово url (), всередині дужок пишеться відносна або абсолютна адреса файлу. При цьому адресу можна задавати в необов'язкових подвійних лапках.

Колір

У стилях можна задавати трьома способами:

- за значенням у шістнадцятковій системі числення (3 набори з двоцифрових чисел, кожне число (від 00 до ff) – інтенсивність відповідно червоного, зеленого і синього). Починається значення з символу «#». Наприклад: #3CB371;
- за назвою (наприклад: white, silver, gray, black, red,orange, yellow, green, blue, navy, purple);
- у форматі RGB (наприклад, `rgb(141, 176, 145)`).

Назви кольорів та їх шістнадцяткові значення можна, наприклад, знайти за адресою https://www.w3schools.com/colors/);

Значеня кольору у форматі RGB задається шляхом задання інтенсивності кожної із складових, «змішуванням» яких одержуємо потрібний колір (червоного, зеленого, синього). Значення кожного з кольорів може змінюватися від 0 до 255. Також можна задавати колір у відсотковому відношенні. Спочатку

вказується ключове слово rgb, а потім в дужках, вказуються компоненти кольору, наприклад rgb (255, 51, 51) або rgb (100%, 20%, 20%).

Приклад

```
body {
 color: red;
}

h1 {
 color: #00ff00;
}

p {
 color: rgb(0,0,255);
}
```

Види селекторів

Як було зазначено раніше, селектори описують ту частину веб-сторінки, до якої необхідно застосувати вказані правила форматування. Можна виділити такі види селекторів:

- універсальний селектор;
- селектор елемента;
- селектор класу;
- селектор ідентифікатора;
- селектор нащадка;
- дочірній селектор;
- сусідній селектор;
- селектор атрибуту;
- селектор псевдокласу;
- селектор псевдоелемента.

Універсальний селектор

Універсальний селектор вибирає усі елементи на сторінці для застосування до них стилів оформлення. Наприклад, наступне правило говорить, що для кожного елемента на сторінці повинна бути додана суцільна чорна границя товщиною в 1 піксель:

Загальний вигляд	*** {** **властивість1:значення1;** … **властивістьN:значення;** **}**
Приклад	***{** **border: 1px solid #000000;** **}**

Селектор елемента

У якості селектора може виступати будь-який тег HTML для якого визначаються правила форматування, такі як: колір, фон, розмір і т.д. Ім'я тега не чутливе до регістра.

Загальний вигляд	ім'я тега **{** **властивість1:значення1;** … **властивістьN:значенняN;** **}**
Приклад	**p {** **text-align: justify;** **color: green;** **}**

У даному прикладі змінюється колір тексту і вирівнювання тексту абзацу. Стиль буде застосовуватися тільки до тексту, який розташовується всередині контейнера <p>.

Слід розуміти, що хоча стиль можна застосувати до будь тегу, результат буде помітний тільки для тегів, які безпосередньо відображаються в контейнері <body>.

Селектор класу

Селектор класу застосовують, коли необхідно визначити стиль для окремих елементів, які логічно поєднані у групи або ж класи. Такі елементи повинні містити атрибут class.

Загальний вигляд	<тег **class** = "Ім'я_класу">
Приклад	<h1 **class="headline"**> Зміст </h1>

Імена класів повинні починатися з латинської літери і можуть містити в собі символ дефіса "-" та підкреслення "_". Використання російських букв в іменах класів є недопустимим. Відмітимо, що елемент може містити декілька класів.

Селектор класу дозволяє задавати стилі для одного або багатьох елементів з однаковим іменем класу.

Загальний вигляд	. Ім'я_класу{
	властивість1:значення1;
	…
	властивістьN:значення;
	}

Приклад	------ У тексті HTML-сторінки -----
	<h1 **class="headline"**> Зміст </h1>
	<p **class="headline"**> Важливий текст </p>
	------ У таблиці стилів CSS ----
	.headline {
	text-transform: uppercase;
	color: lightblue;
	}

Можна також звузити діапазон елементів, які буде вибирати селектор вказавши ім'я елемента (тег), до якого буде застосовуватися форматування. Для цього спочатку вказують ім'я тегу і потім через крапку (без пробілів) вказують ім'я класу.

Загальний вигляд	**тег**.ім'я_класу{
	властивість1:значення1;
	…
	властивістьN:значенняN;
	}

Приклад	------ У тексті HTML-сторінки ----- <h1 **class="headline"**> Зміст </h1> ------ У таблиці стилів CSS ---- **h1.headline** { text-transform: uppercase; color: lightblue;

	}

Селектор ідентифікатора

Ідентифікатор (або ж "id") визначає унікальне ім'я елемента на сторінці, яке використовується для зміни його стилю і звернення до нього через скрипти. Ідентифікатор елемента задається за допомогою атрибуту id.

Загальний вигляд	<тег **id** = "ідентифікатор">
Приклад	------ У тексті HTML-сторінки ----- <p **id="first_line"**> Зміст </p>

Ідентифікатор повинен починатися з символу латинського алфавіту і може містити в собі символ дефіса (-) та підкреслення (_). Використання російських букв в іменах ідентифікатора є недопустимим.

Селектор вибору елемента за його id починається з символу "#", потім іде ім'я ідентифікатора.

Загальний вигляд	# **id** елемента **{** властивість1:значення1; … властивістьN:значенняN; **}**

Приклад	------ У тексті HTML-сторінки -----
	<p **id="first_line"**> Зміст </p>
	------ У таблиці стилів CSS ----
	#first_line {
	text-transform: uppercase;
	color: lightblue;
	}

Періодично піднімається суперечка про доцільність використання ідентифікаторів у верстці. Насправді немає різниці, через що задавати стилі, але слід пам'ятати про деякі особливості ідентифікаторів і класів.

Ідентифікатори (id)	**Класи**
у коді документа кожен ідентифікатор унікальний і повинен бути включений лише один раз	клас може використовуватися в коді багато разів для різних елементів
ім'я ідентифікатора чутливе до регістру	імена класів чутливі до регістра
стиль для ідентифікатора має вищий пріоритет, ніж у класів	
значення ідентифікатора визначається як одна величина	класи можна комбінувати між собою, додаючи кілька класів до одного тегу і розділяючи їх пробілом

Селектор нащадків

Селектори нащадків використовують у тому випадку, якщо потрібно вказати на елементи (нащадки), які знаходяться всередині інших (предків). Селектор при цьому складається з послідовності тегів розділених пропусками (пробілами чи символами tab), у якій кожен наступний є нащадком попереднього (предок нащадок1 нащадок_нащадка1 . . . { правила форматування }).

Елементи у HTML документі	<тег_1> . . . . **<тег_2>** ... вміст тегу ... **</тег_2>** . . . **<тег_2>** ... вміст тегу ... **</тег_2>** . . . </тег_1>

Селектор вибору	тег_1 тег_2 {
	властивість1:значення1;
	…
	властивістьN:значенняN;
	}

Приклад	------ У тексті HTML-сторінки -----
	<section>
	<h2> Підрозділ 1 </h2>
	<p>
	. . . Текст підрозділу 1 . . .
	</ p >
	<h2> Підрозділ 2 </h2>
	< p >
	. . . Текст підрозділу 1 . . .
	</ p >
	</section >
	------ У таблиці стилів CSS ----
	/*Вибрати усі елементи-нащадки <**p**>, які знаходяться всередині тегів-предків <**section**> */
	section p{
	text-transform: uppercase;
	color: lightblue;
	}

При цьому теги-предки не обов’язково повинні бути безпосередніми предками, головне, щоб якийсь із їх предків був той, який зазначений у селекторі.

Приклад.

```
------ У тексті HTML-сторінки -----
```

```
<section>
 <h2> Підрозділ 1 </h2>
     <p>
      . . . Текст підрозділу 1 . . .
     </p>
 <h2> Підрозділ 2 </h2>
 <div>
 . . .
          <p>
           . . . Текст підрозділу 1 . . .
          </p>
 </div>
</section >
------ У таблиці стилів CSS ----
/*Вибрати усі елементи-нащадки <p>, які знаходяться всередині
тегів-предків < section > */
section p{
 text-transform: uppercase;
 color: lightblue;
}
```

Зауважимо, що предків і нащадків можна задавати і за допомогою селекторів класів та ідентифікаторів

Предок є елементом певного класу	1) Усі теги <p>, які знаходяться всередині елемента з класом “info”
	`.info p{ color:red }`
	2)Усі теги <p>, які знаходяться всередині тегів <section> з класом “info”
	`section.info p{ color:red }`
	3) Усі теги <p>, які знаходяться всередині тегів з класом “info”, а вони в свою чергу знаходяться всередині тегу <section> (між тегом і класом вставлено пропуск)
	`section .info p{ color:red }`
	3) Усі теги з класом “hot”, які знаходяться всередині тегів з класом “news”
	`.news .hot{ color:red }`

Предок є елементом	Елементи <p>, які знаходяться всередині тегу із вказаним id=“**intro**”

із заданим id	**#intro** p{
	color:red
	}

Зауважимо, що можна також використовувати і універсальний селектор “*”. Символ “*” при цьому вказує на те, що потрібна наявність деякого елемента.

Правило	Елементи, для яких буде застосовано
p *{ color:red } **Усі теги всередині тегів p**	 The ***main* ** part - ← відповідає тегу “p” The ** *main*** ← відповідає “*” ***important*** **** part

p * *{
color:red
}
 	
Усі елементи, які знаходяться всередині деякого тегу, який знаходиться всередині тегу “p”
	The ← відповідає тегу “p”
	 ← відповідає першій “*”
	main
	**** ← відповідає другій “*”
	important
	****
	
	part

p * a{
color:red
}
 	
Усі теги “a”, які знаходяться всередині деякого тегу, який знаходиться всередині тегу “p”
	The
	 main
	important
	part of
	information
	
	part
	information2

Дочірній селектор

Іноді потрібно вказати на елемент, яким є безпосереднім нащадком заданого предка (тобто вказаний нащадок знаходиться всередині вказаного предка, але він не знаходиться в середині ще якихось тегів). В цьому випадку між предком і нащадком ставлять знак "**>**"

Загальний вигляд	предок > нащадок{
	. . . правила форматування . . .
	}

Приклад	------ У таблиці стилів CSS ----
	Усі теги , для яких тег <section> є безпосереднім предком
	section **>** p{
	color:red
	}
	
	------ У тексті HTML-сторінки -----
	<section>
	  Підрозділ 1
	** ← *Підходить***
	. . . Текст підрозділу 1 . . .
	** **
	  Підрозділ 2
	
	. . .
	 ← ***Даний*** ***не підходить бо не є безпосереднім нащадком*** **<section>**
	. . .
	
	</section >

Селектор сусідів

Селектор сусідів застосовують у випадку, коли необхідно вказати (вибрати для форматування) на елементи, для яких предок є тим же самим, як і для деяких конкретно вказаних елементів (базових елементів). Для цього у селекторі вибору необхідно вказати селектор базового елемента та селектор потрібного сусіда, розділивши їх знаком "+" або знаком "~".

При використанні знаку "+" дія селектора поширюється тільки на безпосереднього сусіда базового елемента, який слідує після базового. Елемент вважається безпосереднім сусідом, якщо між базовим елементом і цільовим сусіднім елементом немає інших тегів. Розглянемо декілька прикладів, де базовим є тег <h1>, а цільовим сусіднім елементом є тег .

Є безпосередніми сусідами	** … ...** допустимий деякий текст без тегів **... … **

Не є безпосередніми сусідами	** … ...** є принаймні один тег...** … **

Наведемо приклад форматування безпосередніх сусідів

	<section> ← **спільний предок** Текст сусіда, але не безпосереднього Деякий довільний текст (можливо і з наявністю тегів) Підрозділ 1.1 Деякий текст БЕЗ ТЕГІВ ** ** **ТЕКСТ БЕЗПОСЕРЕДНЬОГО СУСІДА** ** ** Деякий довільний текст (можливо і з наявністю тегів) Текст сусіда, але не безпосереднього Деякий довільний текст (можливо і з наявністю тегів) Текст сусіда, але не безпосереднього Підрозділ 1.2 ** ** **ТЕКСТ БЕЗПОСЕРЕДНЬОГО СУСІДА** ** ** </section>

	h1+p{
	 color:red
	}

Якщо потрібно вибрати усіх сусідів, які розташовані після базового, то замість знаку "+" потрібно використовувати знак "~".

Приклад.

	<section> ← **спільний предок** ← не підходить, бо знаходиться перед базовим Текст сусіда, але не безпосереднього Деякий довільний текст (можливо і з наявністю тегів) Підрозділ 1.1 ← базовий Деякий текст БЕЗ ТЕГІВ ** ** **ТЕКСТ БЕЗПОСЕРЕДНЬОГО СУСІДА**

	** ** Деякий довільний текст (можливо і з наявністю тегів) ** ** **Текст сусіда, але не безпосереднього** ** ** Деякий довільний текст (можливо і з наявністю тегів) ** ** **Текст сусіда, але не безпосереднього** ** ** Підрозділ 1.2 ← базовий ** ** **ТЕКСТ БЕЗПОСЕРЕДНЬОГО СУСІДА** ** ** </section>

	h1~p{
	color:red
	}

Селектор атрибуту

Селектор атрибуту використовують у випадку, коли необхідно здійснити вибірку елементів, для яких:

- визначено певний атрибут (значення атрибуту не є важливим);
- значення атрибуту співпадає із заданим;
- вказане значення є частиною значення, яке вказано в атрибутах;
- значення атрибуту починається із заданої частини;
- значення атрибуту закінчується заданим фрагментом;
- значення атрибуту зустрічається у якомусь місці;
- значення атрибуту містить вказаний фрагмент як ціле слово, можливо відділене від інших пробілом.

Визначено певний атрибут (значення атрибуту не є важливим)

Дане правило може формуватися для усіх тегів, які мають вказаний атрибут або ж тільки для конкретно заданих тегів, які містять заданий атрибут

	Загальний вигляд	Приклад

Для будь-яких тегів, які містять атрибут	[назва атрибуту]
	{
	правила форматування
	}
		[**id**]{
		color:red
		}
		
		Усі теги з атрибутом "id"

Тільки для заданих тегів, які	назва тегу [назва атрибуту]
	{
	правила форматування
		p[**id**]{
		color:red
		}

містять вказаний атрибут	}	**Усі теги “p” з атрибутом “id”**

Розглянемо ще декілька прикладів

Приклади задання тільки назви атрибуту

Загальний вигляд	[назва атрибуту]
	{
	правила форматування
	}

Приклад	- - - - - - - Правило CSS - - - - - - - -
	[id]{
	color:red
	}
	- - - - - - - У файлі HTML - - - - - - - -
	 First
	<p **id= “middle”**>Second ←Підходить бо має атрибут id
	 Third
	<a **id= “lk”** href=”#”>Link ←Підходить бо має атрибут id

Приклад	- - - - - - - Правило CSS - - - - - - - -
	[class]{
	color:red
	}
	- - - - - - - У файлі HTML - - - - - - - -
	<p **class= “start”**>First ←Підходить бо має атрибут class
	 Second
	 Third
	<p **class= “last”**>End ←Підходить бо має атрибут class

Приклади задання тегів із заданим атрибутом

Загальний вигляд	назва тегу [назва атрибуту]
	{
	правила форматування
	}

Приклад	- - - - - - - Правило CSS - - - - - - - -
	p[id]{
	color:red
	}
	- - - - - - - У файлі HTML - - - - - - - -
	 First
	<p **id= “middle”**>Second ←Підходить, бо це “p” із “id”
	 Third
	Link

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	em[class]{
	 color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	 First
	 Second
	 Third
	<em **class= “last”**>*End* ←Підходить, бо це “em” з “class”

Значення атрибуту співпадає із заданим

Аналогічно до попереднього випадку можливі два випадки: коли вказано лише значення атрибуту і коли зазначено також тег.

	Загальний вигляд	Приклад

Для будь-яких тегів, які містять атрибут з вказаним значенням	[атрибут=“значення”]
	{
	 правила форматування
	}
		[class=“info”]{
		 color:red
		}

Тільки для заданих тегів, які містять вказаний атрибут із зазначеним значенням	тег [атрибут= “значення”]
	{
	 правила форматування
	}
		p[class=“info”]{
		 color:red
		}

Розглянемо декілька прикладів:

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	[class= “start”]{
	 color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	<p **class= “start”**>First ←Підходить
	<p class= “start middle”>Second
	 Third
	<a **class = “start”** href=”#”>Link ←Підходить

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	p[class= “start”]{
	 color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	<p **class= “start”**>First ←Підходить
	<p class= “start middle”>Second
	 Third
	Link

Значення атрибуту починається із заданої частини

	Загальний вигляд	Приклад

Для будь-яких тегів, для яких значення атрибуту починається із вказаного фрагменту	[атрибут **^=** “значення”]
	{
	правила форматування
	}
		[**class^=“info”**]{
		color:red
		}

Тільки для заданих тегів, для яких значення атрибуту починається із вказаного фрагменту	тег [атрибут **^=** “значення”]
	{
	правила форматування
	}
		p[**class^=“info”**]{
		color:red
		}

Розглянемо приклади

Приклад	- - - - - - - Правило CSS - - - - - - - -
	[class^= “start”]{
	color:red
	}
	- - - - - - - У файлі HTML - - - - - - - -
	<p **class= “start”**>First ←Підходить
	 Second
	<p class= “middle_info_ start”> Third
	<a **class = “startup”** href=”#”>Link ←Підходить

Приклад	- - - - - - - Правило CSS - - - - - - - -
	p[id^= “start”]{
	color:red
	}
	- - - - - - - У файлі HTML - - - - - - - -
	 First
	 Second
	 Third ←Підходить
	Link ←Підходить

Значення атрибуту закінчується заданим фрагментом

	Загальний вигляд	Приклад

Для будь-яких тегів, для яких значення атрибуту закінчується вказаним фрагментом	[атрибут **$=** “значення”]
	{
	правила форматування
	}
		[**class $= “info”**]{
		color:red
		}

Тільки для заданих тегів, для яких значення атрибуту закінчується вказаним фрагментом	тег [атрибут **$=** “значення”]
	{
	 правила форматування
	}
		p[**class $=** **“info”**]{
		 color:red
		}

Розглянемо приклади

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	[class $= “start”]{
	 color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	 First
	<p **class= “middle_zonestart”**>Second ←Підходить
	<p **class= “middle-info-start”**> Third
	←Підходить
	Link

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	p[id $= “start”]{
	 color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	<p **id= “part-start”**>First ←Підходить
	 Second
	 Third
	<a **id = “my_first_start”** href=”#”>Link
	←Підходить

Значення атрибуту містить вказаний фрагмент у якомусь довільному (на початку, всередині, в кінці) місці

	Загальний вигляд	Приклад

Для будь-яких тегів, для яких значення атрибуту містить вказаний фрагмент (будь-де)	[атрибут ***=** “значення”]
	{
	 правила форматування
	}
		[**class** ***=** **“info”**]{
		 color:red
		}

Тільки для заданих тегів, для яких значення атрибуту містить вказаний фрагмент (будь-де)	тег [атрибут ***=** “значення”]
	{
	 правила форматування
	}
		p[**class** ***=** **“info”**]{
		 color:red
		}

Розглянемо приклади

Приклад	- - - - - - - - Правило CSS - - - - - - - - -

	[class *= "start"]{
	color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	<p **class= "start"**>First ←Підходить
	<p **class= "middle_start_zone"**>Second
	←Підходить
	<p **class= "middle-info-start"**> Third
	←Підходить
	<a **class = "startup"** href="#">Link ←Підходить

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	p[id *= "start"]{
	color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	<p **id= "part-start"**>First ←Підходить
	 Second
	<p **id = "start_info"**> Third ←Підходить
	Link

Значення атрибуту містить вказаний фрагмент як ціле слово, можливо відділене від інших пробілом

Даний селектор використовується у випадку, коли значення атрибуту може складатися з декількох слів розділених пробілом (наприклад класи)

	Загальний вигляд	Приклад

Для будь-яких тегів, для яких значення атрибуту містить вказане слово (можливо відділене пробілом)	[атрибут **~=** "значення"]
	{
	правила форматування
	}
		[**class ~= "info"**]{
		color:red
		}

Тільки для заданих тегів, для яких значення атрибуту містить вказане слово (можливо відділене пробілом)	тег [атрибут **~=** "значення"]
	{
	правила форматування
	}
		p[**class ~= "info"**]{
		color:red
		}

Розглянемо приклади

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	[class ~= "start"]{
	color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	<p **class= "start"**>First ←Підходить

	Second <p **class= "info start"**> Third ←Підходить <a **class = "base start info"** href="#">Link ←Підходить

Приклад	- - - - - - - Правило CSS - - - - - - - -
	p[class ~= "start"]{
	color:red
	}
	- - - - - - - У файлі HTML - - - - - - - -
	<p **class= "part start"**>First ←Підходить
	 Second
	 Third
	Link

Значення атрибуту співпадає із заданим фрагментом або починається з вказаного фрагменту як ціле слово, відділене від інших дефісом

Даний селектор використовується у випадку, коли значення атрибуту може складатися з декількох слів розділених дефісом (наприклад класи чи ідентифікатори)

	Загальний вигляд	Приклад

Для будь-яких тегів, для яких значення атрибуту починається з вказаного фрагменту (можливо відділене дефісом)	[атрибут **|=** "значення"]
	{
	правила форматування
	}
		[**class |= "info"**]{
		color:red
		}

Тільки для заданих тегів, для яких значення атрибуту починається з вказаного фрагменту (можливо відділене дефісом)	тег [атрибут **|=** "значення"]
	{
	правила форматування
	}
		p[**class |= "info"**]{
		color:red
		}

Розглянемо приклади

Приклад	- - - - - - - Правило CSS - - - - - - - -
	[class |= "start"]{
	color:red
	}
	- - - - - - - У файлі HTML - - - - - - - -
	<p **class= "start"**>First ←Підходить
	 Second

	Third <a **class = "start-base info"** href="#">Link ←Підходить

Приклад	- - - - - - - - Правило CSS - - - - - - - - -
	p[id |= "start"]{
	 color:red
	}
	- - - - - - - - У файлі HTML - - - - - - - - -
	 First
	 Second
	<p **id = "start-info"**> Third ←Підходить
	Link

Селектор псевдокласу

Селектори псевдокласів не відносяться до якихось наперед визначених елементів і визначають динамічний стан елементів, який:

- змінюється з часом;
- змінюється користувачем;
- змінюється положення у дереві документа.

Прикладом такого стану служить текстове посилання, яке змінює свій колір при наведенні на неї курсора миші. При використанні псевдокласів браузер не перевантажує поточний документ, тому за допомогою псевдокласів можна отримати різні динамічні ефекти на сторінці.

Синтаксис псевдокласів

Псевдокласи можуть застосовуватися як з використанням додаткового селектору (відділяємо селектор від псевдокласу двокрапкою), так і без нього.

	Загальна форма	Приклад

з використанням додаткового селектора	селектор : псевдоклас {
	 Опис правил стилю
	}
		A.menu **: hover** {
		 color: green
		}

без використання додаткового селектора (правило буде застосовано до усіх елементів)	: псевдоклас {
	 Опис правил стилю
	}
		: hover {
		 color: green
		}

Умовно всі псевдокласи діляться на три групи:

- визначають стан елементів;
- мають відношення до дерева елементів;
- вказують мову тексту.

Псевдокласи, які визначають стан елементів

До цієї групи відносяться псевдокласи, які розпізнають поточний стан елемента і застосовують стиль тільки для цього стану.

Для посилань

	Для чого застосовується	**Приклад**

:link	Застосовується до невідвіданих посилань, тобто таких, на які користувач ще не натискав	a:**link** {
		color:brown
		}

:visited	Застосовується до відвіданих посилань. Зазвичай таке посилання змінює свій колір за замовчуванням на фіолетовий, але за допомогою стилів колір і інші параметри можна задати самостійно.	a:**visited** {
		color:red
		}

Браузер деякий час зберігає історію відвідувань, тому посилання може бути позначене як відвідане хоча б тому, що по ньому був зафіксований перехід раніше. Записи `a{...}` і `a:link{...}` за своїм результатом рівноцінні, оскільки в браузері дає один і той же ефект, тому псевдоклас: `link` можна не вказувати. Винятком є якорі, на них дія: `link` не поширюється.

Стан елементів

	Загальна форма	Приклад

:active	Відбувається при активації користувачем елементу. Наприклад, посилання стає активним, якщо навести на нього курсор і клацнути мишкою. Незважаючи на те, що активним може стати практично будь-який елемент сторінки, псевдоклас: active використовується переважно для посилань	a: **active**{
		color:green
		}

: focus	Застосовується до елемента при отриманні ним фокусу. Наприклад, для текстового поля форми отримання фокусу означає, що курсор встановлений в полі, і за допомогою клавіатури можна вводити в нього текст. Результат буде видно тільки для тих елементів, які	a: **focus**{
		color:red
		}

	можуть отримати фокус. Зокрема, це <a>, <input>, <select> і <textarea>.	
:checked	виділені (вибрані користувачем) елементи форми	**:checked{** color:blue **}**

:hover	активізується тоді, коли курсор миші знаходиться в межах елемента, але клацання по ньому не відбувається. Приклад таблиці, яка виділяє рядок, на який наведено курсор.	tr**:hover**{
		background: #fc0;
		}

Доступність для зміни елемента

	Загальна форма	Приклад

:enabled	всі доступні для зміни (не заблоковані) елементи форми	:enabled{
		color:green
		}

:disabled	недоступні для зміни (заблоковані) елементи форми	:disabled{
		color:green
		}

Коректність заповнення полів форми

	Загальна форма	Приклад

:valid	поля форми, які заповнені коректно (пройшли перевірку у браузері на відповідність визначеним правилам)	**:valid**{
		color:green
		}

:invalid	поля форми, значення яких не відповідають визначеним для них правилам	**:invalid**{
		color:green
		}

:in-range	поля форми, значення яких знаходяться у заданому діапазоні	**:in-range** {
		color:green
		}

:out-of-range	поля форми, значення яких не знаходяться у заданому діапазоні	**: out-of-range** {
		color:red
		}

Особливі властивості елемента у документі

	Загальна форма	Приклад

:target	Адреса у вигляді – "https://website.ua/forum#**first**" вказує на певний елемент на сторінці з	**:target**{
		color:green
		}

	ідентифікатором first (id="first"). Цей елемент називається цільовим. Селектор **:target** обирає цей цільовий елемент.	

:lang	За допомогою псевдокласів можна змінювати стиль оформлення іноземних текстів, а також деякі налаштування. За допомогою псевдокласу **:lang** можна задавати певні налаштування, характерні для різних мов, наприклад, вид лапок в цитатах. В якості мови можуть виступати наступні значення: en - англійська; de - німецька; ru - російська; fr - французька; it – італійська та ін.	**:lang(en)** {
		color:blue
		}

:not(селектор)	елементи, які не відповідають вказаному селектору	**:not([type="submit"])**{
		color:blue
		}

Селектори структурних псевдоелементів

	Загальна форма	Приклад

:first-child	вибирає тільки перший дочірній елемент	**:first-child**{
		color:blue
		}

:last-child	вибирає тільки останній дочірній елемент	**:last-child**{
		color:blue
		}

:only-child	вибирає дочірні елементи тільки у випадку, якщо вони є єдиними у їх батьківському елементі	**:only-child**){
		color:blue
		}

:nth-child(odd)	вибирає непарні дочірні елементи	:nth-child(odd){
		color:blue
		}

:nth-child(even)	вибирає парні дочірні елементи	:nth-child(even){
		color:blue
		}

:nth-child(номер)	вибір дочірнього елемент з вказаним **номером**	**:nth-child(3){** color:blue **}**

		третій дочірній
:nth-child(n+ номер)	вибір усіх дочірніх елементів починаючи з вказаного **номера**	**:nth-child(n+2)**{ color:blue } **кожен починаючи з другого**
:nth-child(число n)	вибір елементів з номерами кратними вказаному **числу**	:nth-child(**3**n)**{** color:blue } **кожен третій**
:nth-child (число n + номер)	вибір елементів з номерами кратними вказаному **числу** починаючи з вказаного початкового **номера**	:nth-child(**3**n+**2**){ color:blue } **кожен третій починаючи з другого**
:nth-last-child (число n + номер)	аналогічний до **:nth-child,** але відлік починається з кінця	**p:nth-last-child(2)** { color:blue } **другий дочірній починаючи відлік з кінця**
:empty	вибирає елементи, у яких немає дочірніх	**:empty{** color:blue }
:root	вибирає елемент, який є кореневим у документі – html	**:root{** color:blue }

Селектор структурних псевдокласів типу

Даний селектор використовують у випадку, коли потрібно врахувати не тільки розташування дочірнього елемента, а і його тип.

	Загальна форма	Приклад
:nth-of-type()	вибирає елементи по аналогії з :nth-child(), але при цьому враховується тип дочірнього елемента	p:nth-of-type(2) **{** color:blue } **Вибирає кожен елемент <p>, що є другим дочірнім <p> елемент свого батька**
:nth-last-of-type()	аналогічно до :nth-of-type() але відлік починається з кінця	p:nth-last-of-type(2) **{** color:blue }

		Вибирає кожен елемент <p>, що є другим дочірнім <p> елемент свого батька з кінця

:first-of-type	вибирає перший дочірній елемент вказаного типу	p:first-of-type{
		color:blue
		}

:last-of-type	вибирає останній дочірній елемент вказаного типу	p:last-of-type{
		color:blue
		}

:only-of-type	вибирає дочірній елемент вказаного типу тільки у випадку, якщо він єдиним дочірнім елементом вказаного типу для свого предка	p:only-of-type{
		color:blue
		}

Селектори псевдоелеметів

Псевдоелементи – це умовні «фіктивні» елементи (для них немає спеціальних тегів), які є частиною існуючих елементів або вводяться в документ додатково. Вони дозволяють задати стиль елементів, не визначених в дереві елементів документа, а також генерувати вміст, якого немає у вихідному коді тексту. У CSS3 при використанні псевдоелементів використовують дві двокрапки (::before), хоча браузери підтримують і стару форму запису з однією двокракою (:before).

Вибірка умовних елементів

	Загальна форма	Приклад
::first-letter	Визначає стиль першого символу в тексті **блокового** елемента, до якого додається	p::first-letter **Обирає першу літеру кожного елементу <p>**
::first-line	Визначає стиль першого рядка **блокового** елемента	p::first-line
::selection	Селектор ::selection	::selection { color: red;

	обирає текст, який користувач виділив. Тільки кілька CSS властивостей можуть бути застосовані до селектору ::selection • color (колір тексту) • background (фон тексту) • cursor (вигляд курсору) • outline (контур навколо тексту)	background: yellow; }

::placeholder	Селектор ::placeholder дозволяє змінювати стильове оформлення тексту підказки. Підсказка задається атрибутом placeholder	input[type="text"]::placeholder
		{
		color:blue
		}

Вставка контенту

Для вставки контенту використовується спеціальна властивість `content`

	Загальна форма	Приклад

::before	додає вказаний контент всередину елемента (перед вмістом цього елемента)	p::before {
		content:"Read this: ";
		}
		---------- 	

		li::before {
		content: "¶ ";
		}
		li {
		list-style: none;
		}
		відміняє станадртні маркери

::after	додає вказаний контент всередину елемента (після вмісту цього елемента)	p::after {
		content:"Read this: ";
		}
		
		q::before {
		content: "«";
		color: blue;
		}
		q::after {
		content: "»";
		color: red;
		}

Псевдоелементи − це умовні фіктивні елементи, які є частиною існуючих елементів або вводяться в документ додатково. Вони дозволяють задати стиль елементів, не визначених в дереві елементів документа, а також генерувати вміст, якого немає у вихідному коді тексту. Псевдоелемент і псевдоклас, точки зору синтаксису − це фактично одне і те ж. Різниця тільки в обраних елементах.
:first-letter Визначає стиль першого символу в тексті елемента, до якого додається.
:first-line. Визначає стиль першого рядка блокового тексту.

При створенні стилю для сайту, коли одночасно використовується багато селекторів, можлива поява повторюваних стильових правил. Щоб не повторювати двічі одні і ті ж елементи, їх можна згрупувати для зручності подання та скорочення коду. Селектори групуються у вигляді списку тегів, розділених між собою комами. В групу можуть входити не тільки селектори тегів, але також ідентифікатори і класи. Загальний синтаксис наступний.

```
Селектор 1, Селектор 2, ..., Селектор N {Опис правил стилю}
```

При такому записі правила стилю застосовуються до всіх селекторів, перерахованих в одній групі.

Зведена таблиця селекторів

Селектор	**Приклад**	**Пояснення до прикладу**
.class	.intro	Вибірка всіх елементів з класом "intro".
#id	#firstname	Вибирає елемент з ідентифікатором id="firstname"
*	*	Обирає всі елементи
element	p	Обирає всі елементи <p>.
elemen1t,element2	div, p	Обирає всі елементи і всі елементи <p>.
element1 element2	div p	Обирає всі елементи <p> в елементі
element1>element2	div > p	Обирає всі елементи <p> для яких батьківським елементом є елемент .
element1+element2	div + p	Обирає елемент <p> який розташовується одразу за елементам .
element1~element2	p ~ ul	Обирає всі елементи які розташовані перед елементом <p>
[attribute]	[target]	Обирає всі елементи, у котрих вказано атрибут target.
[attribute=value]	target="_blank"	Обирає всі елементи з target="_blank"
[attribute~=value]	[title~=flower]	Обирає всі елементи з атрибутом title в

Селектор	**Приклад**	**Пояснення до прикладу**
		яких міститься слово flower
[attribute|=value]	[lang|=en]	Обирає всі елементи з атрибутом lang значенням якого починається з "en".
[attribute^=value]	a[href^="https"]	Обирає всі елементи <a> в яких значення атрибута href починається з "https".
[attribute$=value]	a[href$=".pdf"]	Обирає всі елементи <a> в яких значення атрибута href закінчується рядком ".pdf"
[attribute*=value]	a[href*="w3schools"]	Обирає всі посилання в яких атрибут href містить слово "w3schools".
:active	a:active	Вибірка активного посилання.
::after	p::after	Додає що-небудь після контенту елемента <p>
::before	p::before	Додає що-небудь перед контентом елемента <p>
:checked	input:checked	Обирає кожний елемент <input> який має значенняі checked.
:disabled	input:disabled	Обирає всі вимкнені елементи введення.
:empty	p:empty	Обирає кожен елемент <p> без тексту та інших HTML елементів.

Селектор	Приклад	Пояснення до прикладу
:enabled	input:enabled	Обирає всі увімкнені елементи введення.
:first-child	p:first-child	Селектор :first-child обирає всі елементи <p>, які є першим нащадком свого батька.
::first-letter	p::first-letter	Обирає першу літеру кожного елементу <p>.
::first-line	p::first-line	Обирає перший рядок кожного елементу <p>
:first-of-type	p:first-of-type	Обирає елемент <p>, якщо він перший елемент свого батька.
:focus	input:focus	Обирає елемент <input>, який є у фокусі
:hover	a:hover	Обирає посилання на яке наведено курсор.
:in-range	input:in-range	Обирає елементи <input> із значенням, що знаходяться в дозволеному діапазоні.
:invalid	input:invalid	Обирає всі елементи input зі значенням, що не пройшло перевірки.
:lang(language)	p:lang(it)	Обирає кожен елемент <p> з атрибутом lang у значенні "it" (Italian)

Селектор	Приклад	Пояснення до прикладу
:last-child	p:last-child	Обирає елемент <p>, який є останнім нащадком свого батька.
:last-of-type	p:last-of-type	Обирає останній елемент свого батька.
:link	a:link	Обирає всі посилання по яким ще не переходили.
:not(selector)	:not(p)	Вибірка елементів, котрі не містять вказаний селектор.
:nth-child(n)	p:nth-child(2)	Обирає елемент <p>, якщо він другий елемент свого батька.
:nth-last-child(n)	p:nth-last-child(2)	Обирає кожен елемент <p>, що є другим нащадком свого батька починаючи відлік з кінця.
:nth-last-of-type(n)	p:nth-last-of-type(2)	Обирає кожен елемент <p>, що є другим елементом <p> свого батька з кінця.
:nth-of-type(n)	p:nth-of-type(2)	Вибирає кожен елемент <p>, що є другим <p> елемент свого батька.
:only-of-type	p:only-of-type	Обирає кожен елемент <p>, що є єдиним елементом <p> такого типу свого батька.
:only-child	p:only-child	Вибирає елемент <p>, якщо він є

Селектор	Приклад	Пояснення до прикладу
		єдиним нащадком свого батька.
:optional	input:optional	Вибірка елементів <input>, що без атрибута "required".
:out-of-range	input:out-of-range	Вибірка всіх <input> елементів у котрих введене значення вийшло за межі дозволеного діапазону.
:read-only	input:read-only	Обирає <input> елементи, котрі з атрибутом "readonly".
:read-write	input:read-write	Вибирає <input> елементи у яких не вказаний "readonly" атрибут.
:required	input:required	Вибірка елементів <input>, що обов'язкові для заповнення.
:root	:root	Вибирає кореневий елемент документа
::selection	::selection	Вибірка виділеного користувачем тексту.
:target	#news:target	Обирає поточний активний елемент, що має значення атрибута id="news"
:valid	input:valid	Обирає всі елементи введення з допустим значенням.
:visited	a:visited	Обирає посилання, по яким вже було здійснено перехід

Селектор	**Приклад**	**Пояснення до прикладу**
:default	input:default	Обирає елемент введення форми, що без задання.
::placeholder	input[type="text"]::placeholder	Змінює стиль підсказки текстового поля <input>.

Спадкування

Спадкуванням називається перенесення правил форматування для елементів, що знаходяться всередині інших. Такі елементи є дочірніми, і вони успадковують деякі стильові властивості своїх батьків, усередині яких розташовуються.

Специфікацією CSS передбачено успадкування властивостей, які відносяться до текстового вміст таких як: `color, font, letter-spacing, line-height, list-style, text-align, text-indent, text-transform, visibility, white-space и word-spacing`.

Властивості, які стосуються форматування блоків, не успадковуються. Це такі властивості як: `background, border, display, float и clear, height и width, margin, min-max-height и -width, outline, overflow, padding, position, text-decoration, vertical-align и z-index`.

Зауважимо, що за потреби, властивість можна успадковувати примусово. Для цього використовують значення `inherit`.

Приклад	Комірка успадковує від рядка (тега tr) властивості границі.
	tr{
	border:inherit;
	}

Розберемо спадкування на прикладі таблиці. Особливістю таблиць можна вважати строгу ієрархічну структуру тегів. Спочатку слідує контейнер <table>, всередині якого додаються теги <tr>, а потім йде тег <td>. Якщо у стилях для селектора TABLE задати колір тексту, то він автоматично встановлюється для вмісту комірок.

```
TABLE {
 color: red; / * Колір тексту * /
 background: # 333; / * Колір фону таблиці * /
 border: 2px solid red; / * Червона рамка навколо
таблиці * /
 }
```

У даному прикладі для всієї таблиці встановлений червоний колір тексту, тому в комірках він також застосовується, оскільки тег <td> успадковує властивості тега <table>. При цьому слід розуміти, що не всі стильові властивості успадковуються. Так, border задає рамку навколо таблиці в цілому, але не навколо комірки. Аналогічно не успадковується значення властивості background. Тоді чому колір фону у комірок в даному прикладі темний, раз він не успадковується? Справа в тому, що у властивості background як значення за замовчуванням виступає transparent, тобто прозорість. Таким чином колір фону батьківського елементу «проглядає» крізь дочірній елемент.

Щоб визначити, успадковується значення стильового властивості чи ні, потрібно зазирнути в довідник властивостей CSS.

Успадкування дозволяє задавати значення деяких властивостей один раз, визначаючи їх для батьків верхнього рівня. Наприклад, потрібно встановити колір і шрифт для основного тексту. Досить скористатися селектором BODY, додати для нього бажані властивості, і колір тексту всередині абзаців та інших текстових елементів поміняється автоматично.

Каскадування

Абревіатура CSS розшифровується як Cascading Style Sheets (каскадні таблиці стилів), де одним з ключових слів виступає «каскад». Під каскадом у даному випадку розуміється одночасне застосування різних стильових правил до елементів документа - за допомогою підключення декількох стильових файлів, спадкування властивостей та інших методів. Щоб в подібній ситуації браузер розумів, яке в підсумку правило застосовувати до елемента, і не виникало конфліктів в поведінці різних браузерів, введені певні пріоритети.

Нижче наведені пріоритети браузерів, якими вони керуються при обробці стильових правил:

- Стиль користувача з додаванням !important.
- Стиль автора з додаванням !important.
- Стиль користувача.
- Стиль автора.
- Стиль браузера.

Найнижчим пріоритетом має стиль браузера − оформлення, яке за замовчуванням застосовується до елементів веб-сторінки браузером. Це оформлення можна побачити в разі «голого» HTML, коли до документа не додається ніяких стилів.

Ключове слово !important грає роль в тому випадку, коли користувачі підключають свою власну таблицю стилів. Якщо виникає протиріччя, коли стиль автора сторінки і користувача для одного і того ж елемента не збігається, то !important дозволяє підвищити пріоритет стилю.

Синтаксис застосування !Important наступний: спочатку пишеться бажана стильова властивість, потім через двокрапку її значення, а в кінці після пробілу вказується ключове слово !important

Загальний вигляд	Властивість: значення !important;

Вирахування специфічності

Для кожного правила браузер визначає специфічність селектора і у випадку застосування до якогось з елементів декількох правил буде використане те, яке має найбільшу специфічність. Значення специфічності розраховується з урахуванням чотирьох складових (чотирьох чисел), які могли бути зазначені при записі селектора вибору:

- вбудований стиль – `1,0,0,0`;
- задання ідентифікатора – `0,1,0,0`;
- задання класу, псевдокласу, атрибуту – `0,0,1,0`;

- задання тегу елемента чи задання псевдоелемента – 0,0,0,1;
- універсальний селектор не враховується.

Якщо при заданні правила було використано декілька складових, то значення специфічності додаються (як чотирицифрові числа).
Приклад. Розрахунок специфічності

```
h1 {color: lightblue;} /* 0, 0, 0, 1*/
em {color: silver;} /* 0, 0, 0, 1*/
h1 em {color: gold;} /*: 0, 0, 0, 1 + 0, 0, 0, 1 = 0, 0, 0,
2*/
div#main p.about {color: blue;} /*: 0, 0, 0, 1 + 0, 1, 0, 0 +
0, 0, 0, 1 + 0, 0, 1, 0 = 0, 1, 1, 2*/
.sidebar {color: grey;} /* 0, 0, 1, 0*/
#sidebar {color: orange;} /* 0, 1, 0, 0*/
li#sidebar {color: aqua;} /* 0, 0, 0, 1 + 0, 1, 0, 0 =
0,1,0,1*/
```

У випадку задання декількох правил застосовується те, у якого специфічність є більшою (розглядаємо специфічність як чотирицифрове ціле число).

Форматування тексту

inherit Успадковує значення батька.
color: колір | inherit Задає колір.
background: [background-attachment || background-color || background-image || background-position || background-repeat] | inherit
background-attachment: fixed | scroll | inherit
fixed- Робить фонове зображення елемента нерухомим.
Scroll Дозволяє переміщатися фону разом з вмістом.
background-color: колір | transparent | inherit
transparent – прозорий фон
background-image: url(файл) | none | inherit – фонове зображення
background-position Задає початкове положення фонового зображення, встановленого за допомогою властивості background-image. Два значення: положення по горизонталі (може бути - left, center, right) і вертикалі (може бути - top, center, bottom). Крім використання ключових слів положення також можна задавати у відсотках, пікселах або інших одиницях. Якщо застосовуються ключові слова, то порядок їх проходження не має значення, при відсотковому записі спочатку задається положення малюнка по горизонталі, а потім, через пропуск, положення по вертикалі.
background-repeat Визначає, як буде повторюватися фонове зображення, встановлене за допомогою властивості background-image. Можна встановити повторення малюнка тільки по горизонталі, по вертикалі або в обидві сторони. Допустимо вказувати два значення, перше ключове слово задає повторення по горизонталі, друге по вертикалі.

no-repeat Встановлює одне фонове зображення в елементі без його повторень, положення якого визначається властивістю background-position Аналогічно no-repeat no-repeat.

repeat Фонове зображення повторюється по горизонталі і вертикалі. Аналогічно repeat repeat.

repeat-x Фоновий малюнок повторюється тільки по горизонталі. Аналогічно repeat no-repeat.

repeat-y Фоновий малюнок повторюється тільки по вертикалі. Аналогічно no-repeat repeat.

font: [font-style||font-variant||font-weight] font-size [/line-height] font-family | inherit
font-style: normal | italic | oblique | inherit

normal Звичайне написання тексту.

italic Курсивне зображення.

oblique Похиле написання. Курсив і похилий шрифт при всій їх схожості не одне і те ж. Курсив − це спеціальний шрифт імітує рукописний, похилий ж утворюється шляхом нахилу звичайних знаків вправо.

font-variant: normal | small-caps | inherit

small-caps – Задає напис капітелем. ПРИКЛАД ТЕКСТУ

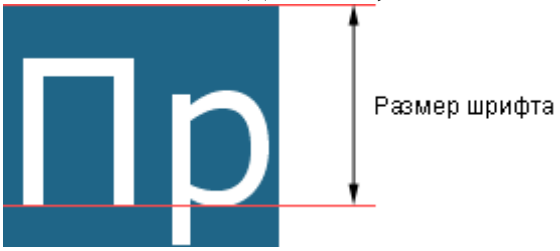
font-weight: bold|bolder|lighter|normal|100|200|300|400|500|600|700|800|900

Нормальне зображення шрифту (яке встановлено за замовчуванням) еквівалентно 400, стандартний напівжирний текст - значенням 700.

font-size Визначає розмір шрифту елемента. Розмір може бути встановлений декількома способами. Набір констант (xx-small, x-small, small, medium, large, x-large, xx-large) задає розмір, який називається абсолютним. По правді кажучи, вони не зовсім абсолютні, оскільки залежать від налаштувань браузера та операційної системи.

Інший набір констант (larger, smaller) встановлює відносні розміри шрифту. Оскільки розмір успадкований від батьківського елементу, ці відносні розміри застосовуються до батьківського елементу, щоб визначити розмір шрифту поточного елементу. Розмір шрифту сильно залежить від значення властивості font-size у батьківського елемента.

Сам розмір шрифту визначається як висота від базової лінії до верхньої межі кегельного майданчика, як показано на рис.



line-height: Встановлює міжрядковий інтервал тексту, відлік ведеться від базової лінії шрифту. За звичайних обставин відстань між рядками залежить від виду та розміру шрифту і визначається браузером автоматично. Від’ємне значення міжрядкового відстані не допускається.

font-family Встановлює сімейство шрифту, яке використовуватиметься для оформлення тексту вмісту. Список шрифтів може включати одну або кілька назв, розділених комою. Якщо в імені шрифту містяться прогалини, наприклад, Trebuchet MS, його необхідно брати в лапки.

Коли браузер зустрічає перший шрифт у списку, він перевіряє його наявність на комп'ютері користувача. Якщо такого шрифту немає, береться наступне ім'я зі списку і також аналізується на присутність. Тому кілька шрифтів збільшує ймовірність, що хоча б один з них буде виявлений на клієнтському комп'ютері. Закінчують список зазвичай ключовим словом, яке описує тип шрифту - serif, sans-serif, cursive, fantasy або monospace. Таким чином, послідовність шрифтів краще починати з екзотичних типів і закінчувати узагальненим ім'ям, яке задає вид накреслення.

Приклад `p { font: bold italic small-caps 12px/14px Forte,"Times New Roman",Serif; }`

text-align: center | justify | left | right | inherit – Вирівнювання тексту

text-decoration: [blink | line-through | overline | underline] | none | inherit – Додає оформлення тексту у вигляді його підкреслення, перекреслення, лінії над текстом і миготіння. Одночасно можна застосувати більше одного стилю, перераховуючи значення через пробіл.

text-transform: capitalize | lowercase | uppercase | none | inherit - Керує перетворенням тексту елемента

capitalize Перший символ кожного слова буде великим. Інші символи свій вигляд не міняють.

lowercase Всі символи перетворюються в нижній регістр. uppercase Всі символи перетворюються у верхній регістр.

text-indent: Встановлює величину відступу першого рядка блоку тексту (наприклад, для абзацу <p>). Впливу на всі інші рядки не відбувається. Допускається від'ємне значення для створення відступу першого рядка, але слід перевірити, щоб текст не виходив за межі вікна браузера.

letter-spacing: Визначає інтервал між символами в межах елемента. Браузери зазвичай встановлюють відстань між символами, виходячи з типу і виду шрифту, його розмірів і налаштувань операційної системи. Щоб змінити це значення і застосовується дана властивість. Допустимо використовувати від'ємне значення, але в цьому випадку треба переконатися, що зберігається читабельність тексту.

word-spacing: Встановлює інтервал між словами. Якщо для тексту задано вирівнювання по ширині через text-align, то властивість word-spacing ігнорується.

list-style: list-style-type || list-style-position || list-style-image | inherit

Комбінації значень повинні слідувати в зазначеному порядку: спочатку йде тип маркера, потім положення і картинка. Жодне значення не є обов'язковим, тому невживані можна опустити.

list-style-type Змінює вигляд маркера для кожного елемента списку. Ця властивість використовується тільки у випадку, коли значення list-style-image

встановлено як none. Маркери розрізняються для маркірованого списку (тег <ul>) і нумерованого (тег <ol>).

Circle Маркер у вигляді кружка.

disc у вигляді точки.

square у вигляді квадрата.

нумерований список

decimal Арабські числа (1, 2, 3, 4, ...).

lower-alpha Рядкові латинські літери (a, b, c, d, ...).

lower-greek Рядкові грецькі літери (α, β, γ, δ, ...).

lower-roman Римські числа в нижньому регістрі (i, ii, iii, iv, v, ...).

upper-alpha Великі латинські літери (A, B, C, D, ...).

upper-roman Римські числа в верхньому регістрі (I, II, III, IV, V, ...).

none Скасовує маркери для списку.

list-style-position: inside | outside

Визначає, як буде розміщуватися маркер щодо тексту. Є два значення: outside - маркер винесений за межі елемента списку і inside - маркер обтікається текстом.

list-style-image: Вказується шлях до зображення-маркеру.

float: left | right | none | inherit

Визначає, по якій стороні буде вирівнюватися елемент, при цьому інші елементи будуть обтікати його з інших сторін. Коли значення властивості float дорівнює none, елемент виводиться на сторінці як зазвичай, при цьому допускається, що один рядок тексту може бути на тій же лінії, що і сам елемент.

Щоб скасувати обтікання, треба після плаваючого елементу розташувати блок з властивістю clear: both;

border: [border-width || border-style || border-color] | inherit

Універсальна властивість border дозволяє одночасно встановити товщину, стиль і колір кордону навколо елемента. Значення можуть йти в будь-якому порядку, розділяючись пробілом, браузер сам визначить, яке з них відповідає потрібному властивості. Для установки кордону тільки на певних сторонах елемента, скористайтеся властивостями border-top, border-bottom, border-left, border-right.

border-width - задає товщину границі. Зазвичай вказується в пікселях, але також можна вказувати ключовими словами thin (2px), medium (4px) і thick (6px).

border-color - визначає колір границі.

border-style - визначає стиль границі. Вони можуть бути наступними:

margin. Встановлює величину відступу від краю елемента. Відступом є простір від кордону поточного елемента до внутрішньої межі його батьківського елементу або до найближчого елемента. Розмірність може бути як в значеннях довжини (px, pt, em і т.д.), так і в значенні auto.

padding встановлюють значення полів навколо вмісту елементу. Полем називається відстань від внутрішнього краю рамки елемента до уявного прямокутника, що обмежує його вміст.

Щоб встановлювати відступи і поля відповідно зверху, справа, знизу, зліва потрібно використовувати ключові слова top, right, bottom, left.

Наприклад: margin-top: 5px; або padding-bottom: 25px;

Значення зовнішніх відступів в деяких випадках може бути і від'ємним.

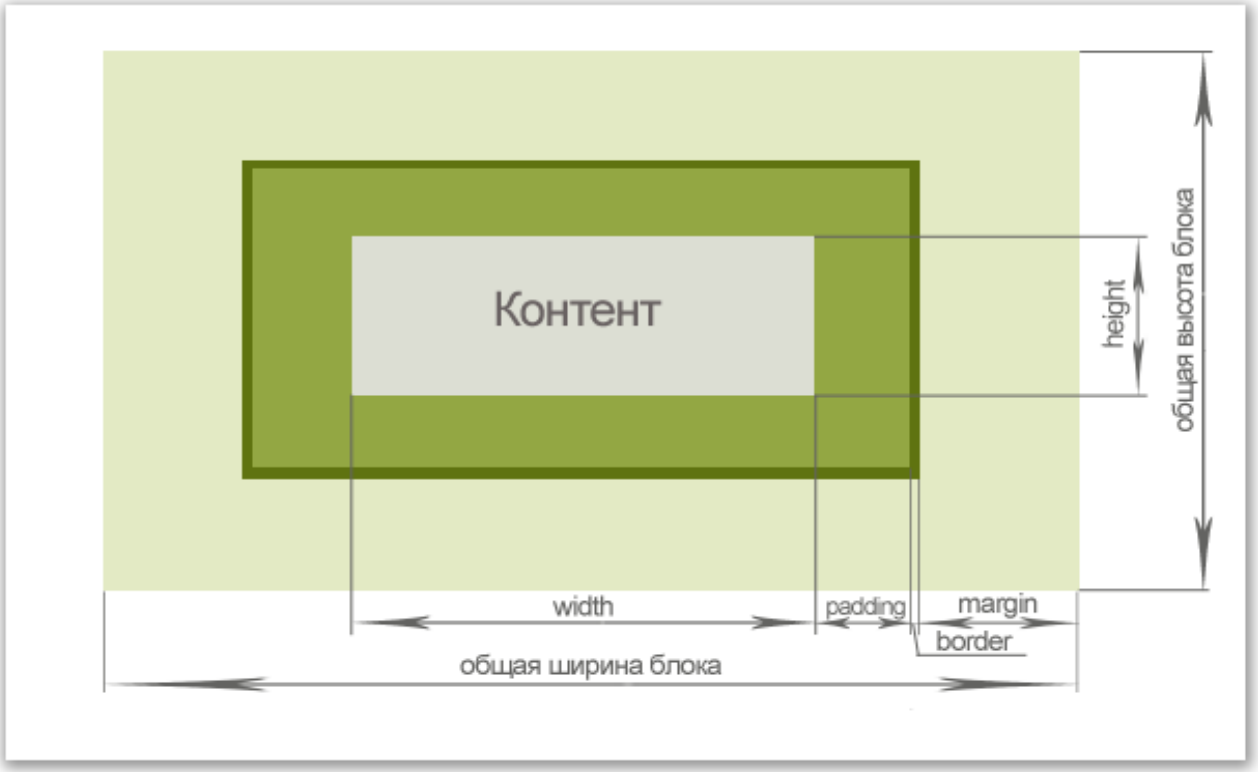
Існує скорочений запис:

margin: 5px; - застосовується до всіх полів;

margin: 5px 10px; - перше число означає відступ знизу і зверху, друге - ліворуч і праворуч;

margin: 5px 10px 4px; - перше число означає відступ зверху, друге - ліворуч і праворуч, третє - знизу;

margin: 5px 10px 4px 6px; - по порядку - зверху, справа, знизу, зліва.



position: absolute | fixed | relative | static | inherit Встановлює спосіб позиціонування елемента щодо вікна браузера або інших об'єктів на веб-сторінці.

absolute Вказує, що елемент абсолютно позиціонується, при цьому інші елементи відображаються на веб-сторінці, немов абсолютно позиціонованого елемента і немає. Положення елемента задається властивостями left, top, right і bottom, також на положення впливає значення властивості position батьківського елементу. Так, якщо в батька значення position встановлено як static або батька немає, то відлік координат ведеться від краю вікна браузера. Якщо у батька значення

position задано як fixed, relative або absolute, то відлік координат ведеться від краю батьківського елементу.

fixed По своїй дії це значення близьке до absolute, але на відміну від нього прив'язується до зазначеної властивостями left, top, right і bottom точці на екрані і не міняє свого положення при прокрутці веб-сторінки.

relative Положення елемента встановлюється щодо його вихідного місця. Додавання властивостей left, top, right і bottom змінює позицію елемента і зрушує його в ту або іншу сторону від первісного розташування.

static Елементи відображаються як зазвичай. Використання властивостей left, top, right і bottom не призводить до якихось результатів.

display: block | inline | inline-block | inline-table | list-item | none | run-in | table | table-caption | table-cell | table-column-group | table-column | table-footer-group | table-header-group | table-row | table-row-group

багатоцільова властивість, яка визначає, як елемент повинен бути показаний в документі.

block Елемент показується як блоковий. Застосування цього значення для вбудованих елементів, наприклад тега , змушує його вести себе подібно блокам − відбувається перенесення рядків на початку і в кінці вмісту.

inline Елемент відображається як вбудований. Використання блокових тегів, таких як і <p>, автоматично створює перенос і показує вміст цих тегів з нового рядка. Значення inline скасовує цю особливість, тому вміст блокових елементів починається з того місця, де закінчився попередній елемент.

inline-block Це значення генерує блоковий елемент, який обтікається іншими елементами веб-сторінки, подібно вбудованому елементу. Фактично такий елемент по своїй дії схожий на вбудовувані елементи (зразок тега <img>). При цьому його внутрішня частина форматується як блоковий елемент, а сам елемент − як вбудований.

inline-table Визначає, що елемент є таблицею як при використанні тега <table>, але при цьому таблиця є вбудованим елементом і відбувається її обтікання іншими елементами, наприклад, текстом.

list-item Елемент виводиться як блоковий і додається маркер списку.

none Тимчасово видаляє елемент із документа. Займане ним місце не резервується і веб-сторінка формується так, немов елемента і не було. Змінити значення і зробити знову видимим елемент можна за допомогою скриптів, звертаючись до властивостей через об'єктну модель. У цьому випадку відбувається переформатування даних на сторінці з обліком знову доданого елемента.

run-in Встановлює елемент як блоковий або вбудований в залежності від контексту.

table Визначає, що елемент є блоковою таблицею подібно використанню тега <table>.

table-caption Задає заголовок таблиці подібно застосуванню тега <caption>.
table-cell Вказує, що елемент являє собою комірку таблиці (тег <td> або <th>).
table-column Призначає елемент колонкою таблиці, немов був доданий тег <col>.
table-column-group Визначає, що елемент є групою однієї чи більше колонок таблиці, як при використанні тега <colgroup>.
table-footer-group Використовується для зберігання однієї або кількох рядків осередків, які відображаються в самому низу таблиці. По своїй дії схоже з роботою тега <tfoot>.
table-header-group Елемент призначений для зберігання однієї або кількох рядків осередків, які представлені вгорі таблиці. По своїй дії схоже з роботою тега <thead>.
table-row Елемент відображається як рядок таблиці (тег <tr>).
table-row-group Створює структурний блок, що складається з декількох рядків таблиці аналогічно дії тега <tbody>.

Список використаної літератури

1. Крис Джамса, Конрад Кинг, Энди Андерсон. Эффективный самоучитель по креативному Web-дизайну. HTML, XHTML, CSS, JavaScript, PHP, ASP, ActiveX. Текст, графика, звук и анимация. Пер с англ. – М.: ООО «ДиаСофтЮП», 2005. – 672 с.
2. Дронов В. А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. – СПб.: БХВ-Петербург, 2011. – 416 с. Режим доступа: http://www.znanium.com/bookread.php?book=351455
3. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. – Питер, 2016. – 768 с.
4. Прохоренок Н. А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. 900 с. Режим доступа: http://www.znanium.com/bookread.php?book=350905
5. Соколов С.А. HTML и CSS в примерах, типовых решениях и задачах. Профессиональная работа. – М.: ИД "Вильямс", 2007. – 416 с.
6. Томсон Л. Разработка Web-приложений на PHP и MySQL: Пер. с англ. – 2-е изд., испр. – СПб: "ДиаСофтЮП", 2003. – 672 с.
7. Ульман Л. Основы программирования на PHP. – М.: ДМК Пресс, 2001. – 288 с.
8. Шмитт Кристофер CSS. Рецепты программирования. – СПб.: "БХВ-Петербург", 2007. – С. 592.
9. Эрик А. Мейер CSS-каскадные таблицы стилей: подробное руководство. – М.: "Символ", 2006. – С. 576.

Зміст

Відповідальний за випуск: завідувач кафедрою системного аналізу і теорії оптимізації
к. ф.-м. н., доц. Кузка О.І.

Автори: к. ф.-м. н., доц. Брила А.Ю.,
к. ф.-м. н., доц. Глебена М.І.,
к. ф.-м. н., доц. Млавець Ю.Ю.,
ст.викл., Ломага М.М.

Рецензенти: к.ф.-м.н., доц. Погоріляк О.О.,
к.т.н., доц. Кондрук Н.Е.

ОСНОВИ CSS
ЧАСТИНА 1

Методичні вказівки для самостійної роботи студентів математичного факультету з дисципліни «Вступ до web-програмування»