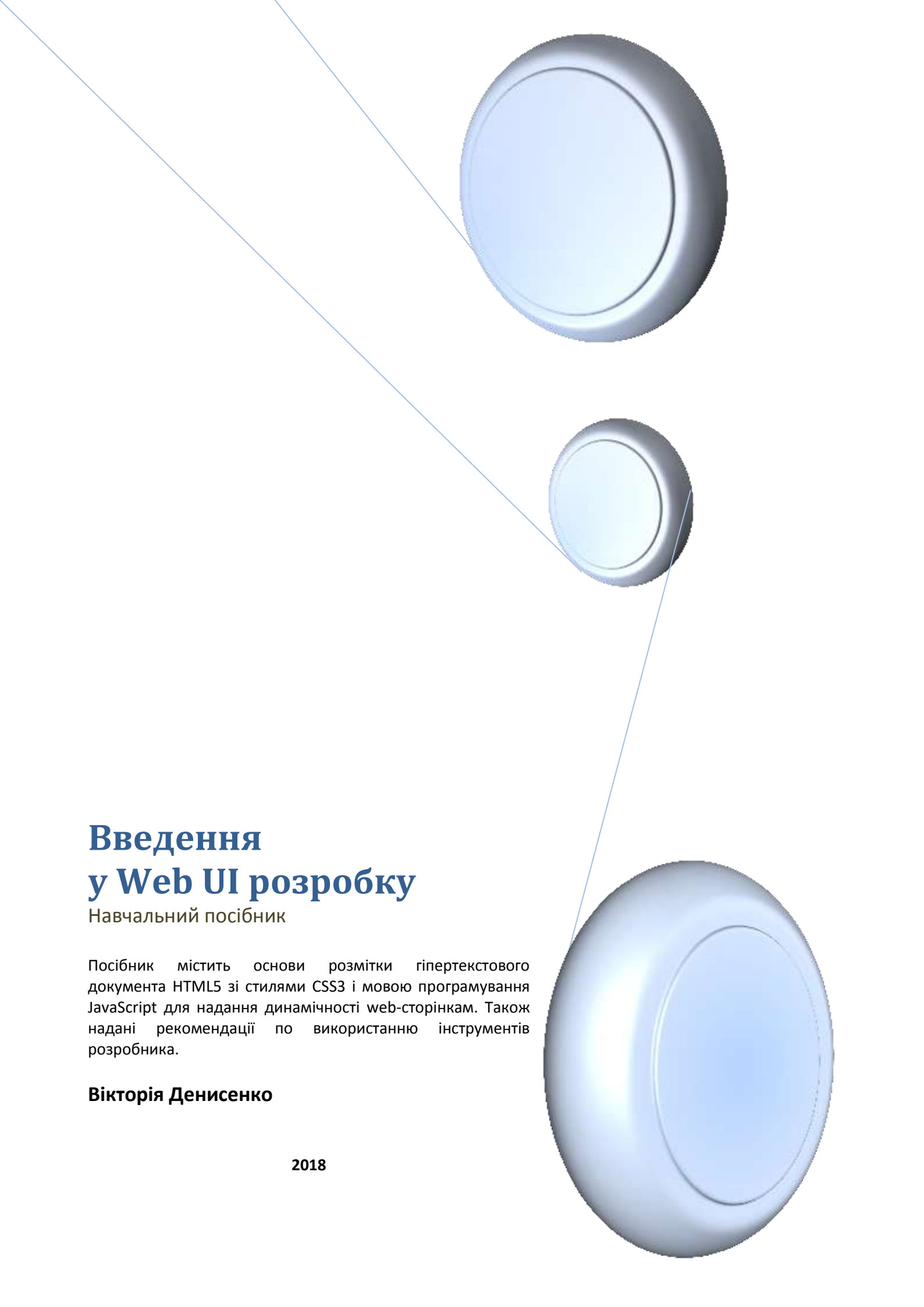


Введення у Web UI розробку

Навчальний посібник

Посібник містить основи розмітки гіпертекстового документа HTML5 зі стилями CSS3 і мовою програмування JavaScript для надання динамічності web-сторінкам. Також надані рекомендації по використанню інструментів розробника.

Вікторія Денисенко

2018

ISBN 978-617-7195-52-7 (PDF)

Міністерство освіти і науки України
Одеська державна академія будівництва та архітектури

«ВВЕДЕННЯ У WEB UI РОЗРОБКУ»

Навчальний посібник

ОДЕСА 2018

УДК 004.42 (075)
Д 33

РЕКОМЕНДОВАНО
Вченою Радою Одеської академії
будівництва та архітектури
протокол № 4 від 28.12.2017р.

Навчальний посібник розглянутий та рекомендований до видання і використання у начальному процесі для судентів освітньої програми «Комп'ютерна механіка», студентів-бакалаврів спеціальності 126 «інформаційні системи та технології», а також аспірантів і викладачів, які виявили інтерес до до прийомів web-програмування, на засіданні кафедри інформаційних технологій та прикладної математики (протокол № 3 від 2.11.2017р.) і на засіданні науково-методичної комісії факультету економіки та управління в будівництві (протокол № 3 від 5.12.2017р.)

Д 33 Денисенко В. Ю.
Введення у Web UI розробку : навч. посіб. / В. Ю. Денисенко. — Одеса : ОДАБА, 2018. — 186 с. : іл. **ISBN 978-617-7195-52-7 (PDF)**

Укладач:

Денисенко Вікторія Юріївна, кандидат технічних наук, доцент ОДАБА

Рецензенти:

Медведєв М.Г. д.т.н., професор, завідувач кафедрою загально-інженерних дисциплін Таврійського національного університету

Кузнєцова Н.В. к.т.н., доцент Інституту прикладного системного аналізу, НТУУ «КПІ ім. Сікорського»

Посібник містить основи розмітки гіпертекстового документа мовою HTML5 зі стилями CSS3 і мовою програмування JavaScript для створення сайтів і надання динамічності web-сторінкам. Також надані рекомендації по використанню інструментів розробника. Наведені тестові завдання для самопідготовки і приклади для самостійного опрацювання розглянутого матеріалу.
Посібник побудовано таким чином, що він може бути корисним, як студентам так і широкому колу споживачів, які виявили інтерес до опанування прийомів web-програмування.

Відповідальна за випуск Лазарєва Д.В., завідувачка кафедрою ІТПМ ОДАБА, канд.техн.наук, доцент

УДК 004.42 (075)
Д 33

ISBN 978-617-7195-52-7 (PDF)

З М І С Т

Вступ

Web-розробка — процес створення web- сайтів або web -додатків. Основними етапами процесу є web - дизайн, верстка сторінок, програмування для web на стороні клієнта (*front end*) і сервера (*back end*), а також конфігурування web - сервера.

Розробка користувальницьких інтерфейсів при грамотному підході будується таким чином, щоб створити його максимально привабливим і зручним для оптимізації його взаємодії з користувачем.

Перед розробниками web-інтерфейсів в будь-якому проекті поставлено завдання створення саме дружнього по відношенню до користувача інтерфейсу. Однак це не завжди таке просте завдання, як може здатися на перший погляд, і часом вимагає чималого досвіду проектування. Головні вимоги тут - зручність, практичність і інтуїтивна зрозумілість. Саме в цей момент вступають в гру такі поняття як UX і UI дизайн.

User Experience Design (UX Design) в перекладі означає «досвід взаємодії» і включає в собі різні UX-компоненти: інформаційну архітектуру, проектування взаємодії, графічний дизайн і контент.

User Interface Design (UI Design) або призначений для користувача інтерфейс - це більш вузьке поняття, яке включає в себе певний набір графічно оформлених технічних елементів.

Правила UI дизайну:

- *Організованість елементів інтерфейсу*. Це означає, що вони повинні бути логічно структуровані і взаємопов'язані.
- *Угруповання елементів інтерфейсу*. Має на увазі об'єднання в групи логічно пов'язаних елементів (меню, форми).
- *Вирівнювання елементів інтерфейсу*. Складно уявити, що погано вирівняний інтерфейс може бути для когось зручним!
- *Єдиний стиль елементів інтерфейсу*. Стильове оформлення грає не останню роль, адже саме воно зберігається в пам'яті користувача.
- *Наявність вільного простору*. Це дозволяє розмежовувати інформаційні блоки, зосереджуючи увагу на чомусь одному.

Розроблений за всіма правилами призначений для користувача інтерфейс значно підвищує ефективність ресурсу і дає йому конкурентні переваги.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними.

Типовим прикладом клієнт-серверної взаємодії є WWW. **W**orld **W**ide **W**eb - всесвітня павутина - служба пошуку та перегляду гіпертекстових документів, що включають в себе графіку, звук і відео. Існує величезна кількість web-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір web-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML.

Але ситуація, як правило, є складнішою; значна частина web-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Розрізняють ***статичні і динамічні web-сторінки***. **Статичні** – це такі сторінки, внести зміни до яких може тільки «творець» web-сторінки. Для створення статичних web-сторінок вистачає мови розмітки HTML з підключеними правилами оформлення CSS.

Динамічні web-сторінки – це сторінки, які можуть бути змінені користувачем сайту. Вміст таких сайтів зберігається не у вигляді статичних HTML сторінок, а знаходиться в базі даних, і відображається «зльоту», безпосередньо за запитом користувача. При створенні динамічних сторінок значна увага приділяється скріптам, які написані мовою JavaScript.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду web-сторінок називаються браузерами (web-оглядачами). Найпоширеніші браузери:

Google Chrome, Firefox, Internet Explorer, Opera і Safari.

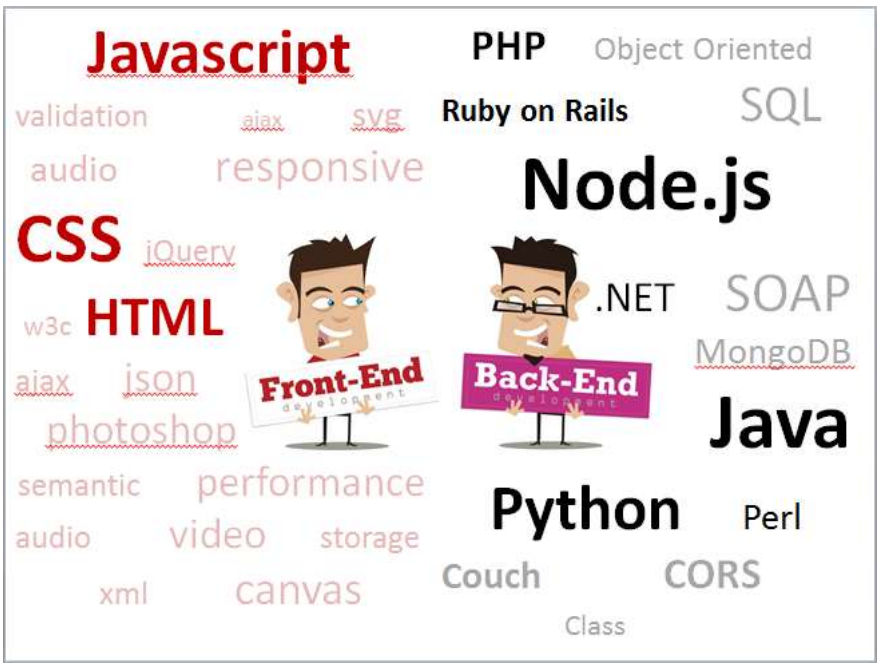
Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме — автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду web-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

В програмній інженерії розрізняють терміни **«front end»** та **«back end»** за принципом розділення відповідальності між рівнем представлення та рівнем доступу до даних відповідно.

Front end — це інтерфейс для взаємодії між користувачем і back end.

Розділення програмних систем на front end та back end спрощує розробку і розділяє підтримку. Емпіричне правило полягає в тому, що front (чи «клієнтська») сторона — це будь-який компонент, яким керує користувач. Код серверної сторони (чи «back end»-у) знаходиться на сервері.

Для програмування front end та back end на сьогодні існує багато мов, але серед професіоналів вирізняють HTML, CSS, JavaScript із бібліотекою jQuery для front end програмування і мови SQL, PHP, Node.js та ін.– для програмування серверів.



Задачі front-end розробника:

- Створення Psd до html / css, оптимізація картинок , пошук шрифтів. Перевірка дизайну. Створення «мокап» і адаптивна розмітка. Комунікація і взаємодія з іншими розробниками.
- Інтеграція з серверною частиною, рефакторинг коду. Пошук оптимальних рішень для того чи іншого завдання. Робота з фреймворками і бібліотеками. Написання юніт тестів. Дизайн архітектури.
- Написання документації. Оптимізація коду SEO / SMO friendly. Постійна комунікація з людьми: замовниками, іншими членами команди, цільовою аудиторією продукту. Безперервний процес навчання.

Основні професії, де задіяні front-end розробники:

- інформаційний архітектор
- web -дизайнер
- верстальник web -сторінок
- програміст
- тестувальник
- пошуковий оптимізатор
- копірайтер (письменник), контент-менеджер

Нерідко web -фахівці поєднують в собі відразу декілька спеціальностей.

Основу створення web-сторінки складає розмітка HTML (**Hyper Text Markup Language**). *Гіпертекст* - інформаційна структура, що дозволяє встановлювати смислові зв'язки між елементами тексту на екрані комп'ютера таким чином, щоб можна було легко здійснювати переходи від одного елемента до іншого. Зараз поняття гіпертексту розширилося до поняття *гіпермедіа*, де елементами інформаційної структури виступають звук, графіка і відео.

Елементи структурної розмітки HTML, які називають *тегі* (tag англійською), застосовуються задля опису семантики тексту, іншими словами ці елементи описують призначення тексту свого контенту. Вони не зазначають ніякого спеціального (візуального) відтворення тексту, проте більшість браузерів мають стандартні стилі форматування для кожного елемента. Для подальшого стилізування тексту рекомендується використовувати Каскадні таблиці стилів (CSS). А для створення динаміки на web- сторінці використовують скріпти мовою JavaScript.

У навчальному посібнику надана інформація, щодо front end розробки, та її основних доданків розробника.

Стандартні стилі браузера і механізм відтворення сторінки, залежить від **«движка»(англ. engine) браузера**. Але слово движок є поширеним в народі, а на професійній мові його називають **CMS** (скорочено від Content Management System), яка дослівно перекладається як система адміністрування сайтами. Тому ще одне визначення «движка» може звучати так - це готова система управління сайтом.

Всього існує 4 «движка»: Trident (для IE), Gecko (для Firefox), Webkit і V8(для Safari, Chrome) Є ще Presto (для Opera), але він практично виходить з професійного поля розробки.

З поняттям «движка» зв'язано і поняття **вендорних префіксів**. Вендорний префікси це приставки, які використовуються виробниками (вендорами) браузерів для експериментальних, ще не прийнятих в стандарт, CSS-властивостей. Наведемо список деяких префіксів для найбільш поширених браузерів:

-webkit- для Chrome і Safari;
-moz- для Firefox;
-ms- для Internet Explorer;
-o- для Opera.

Всі новини у світі Web розробки дивіться на сайті uptodate.frontendrescue.org .

Автор висловлює велику подяку колективу українського громадського проекту масових відкритих on-line- курсів «*Prometheus*», ідеї і матеріали якого лежать у підґрунті створення даного посібника.

Основи HTML

HTML (*HyperText Markup Language*) - це мова розмітки гіпертексту. Вона використовується для створення web-сторінок. Кожна web- сторінка, яку бачить користувач у браузері, є звичайним текстовим файлом, який розроблений за правилами мови HTML.

HTML не є мовою програмування, але «мову розмітки» слід знати для розробки сайтів.

Для написання html- коду достатньо використати будь-який текстовий редактор, навіть Блокнот (Notepad), але існують і професійні засоби розробника, до яких відноситься доданок **Sublime Text 3**. Саме цей доданок ми і будемо використовувати під час знайомства із HTML. Цей доданок може бути завантажений на власний комп'ютер з сайту https://www.sublimetext.com/3 для ознайомлення. Документацію доданка можна подивитися на сайті http://sublimetext.ru/documentation

Для відображення розробленої web- сторінки треба скористатися будь-яким браузером Internet Explorer, Mozilla Firefox, Google Chrome, Opera, Safari (ми у наших вправах будемо використовувати Chrome).

Щоб створити web- сторінку, потрібно створити в файловій системі текстовий файл з розширенням html, або htm. Обидва розширення рівноцінні, htm використовувався тоді, коли за стандартом 8.3 не можна було мати у розширенні більш, ніж 3 символу. Зараз такого обмеження не існує, і ми будемо надавити своїм файлам розширення html.

Серед всіх html- файлів сайту виділяють основний, саме він індексується пошукувачами. Такий файл має назву **index.html**.

Базова структура html-коду

Html-код – це текст, який містить спеціальну розмітку із використанням *тегів*.
Тег (tag) – це конструкція з «кутових дужок», які на клавіатурі набираються, як знаки «<» (менше) і «>» (більше), *наприклад*, <html> <h1> <a>.
Тегі є ***парні***, коли конструкція складається із відкритого (початкового) тегу і закритого (завершує дію по відкритому), *наприклад*,
<html>…</html>
<h1>…</h1>
Данні, які будуть відображатися на сторінці, розташовані поміж відкритим і закритим тегами, *наприклад*
<h1>Заголовок тексту</h1>
Якщо вміст значний, то варто у коді записувати тегі на окремих строках,

наприклад,

```
<p>
    На жаль, браузери відкидають всі символи переносу рядка, табуляції і навіть зайві
    пропуски. Тому віршик буде відображатися в один або кілька рядків (залежно від
    ширини вікна). Але вихід завжди є.
</p>
```

Відкриваючі і закриваючі теги завжди ходять парами. Але існують і непарні тегі, які складаються тільки з відкриваючого тегу. Такі тегі називають *одинарними*.

За сучасним стандартом **всі тегі треба писати «маленькими» строковими літерами.**

Атрибут, якщо він є, може міститися **тільки у відкритому тегі.**

У html-код можна додати *коментар* - текст, який не буде відображений на сторінці. Для цього треба записати:

```
<!-- тут коментар -->
```

Найменший html-код має вигляд

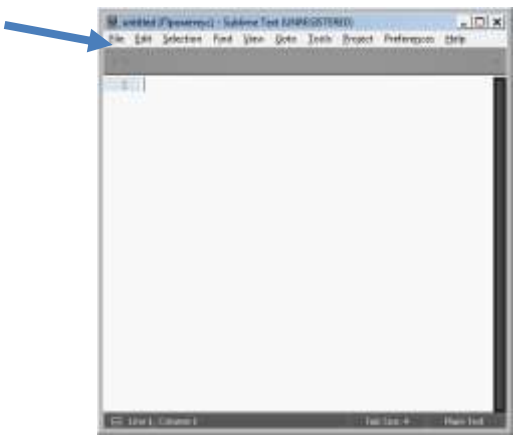
```
<html>
  <head>
    <!--Заголовок сторінки, наприклад: -->
    <title>Моя сторінка</title>
  </head>
  <body>
    <!-- Вміст сторінки -->
  </body>
</html>
```

Структура коду складається із елемента <head></head>, який містить управляючу інформацію. І елементу <body></body>, що містить саму сторінку.

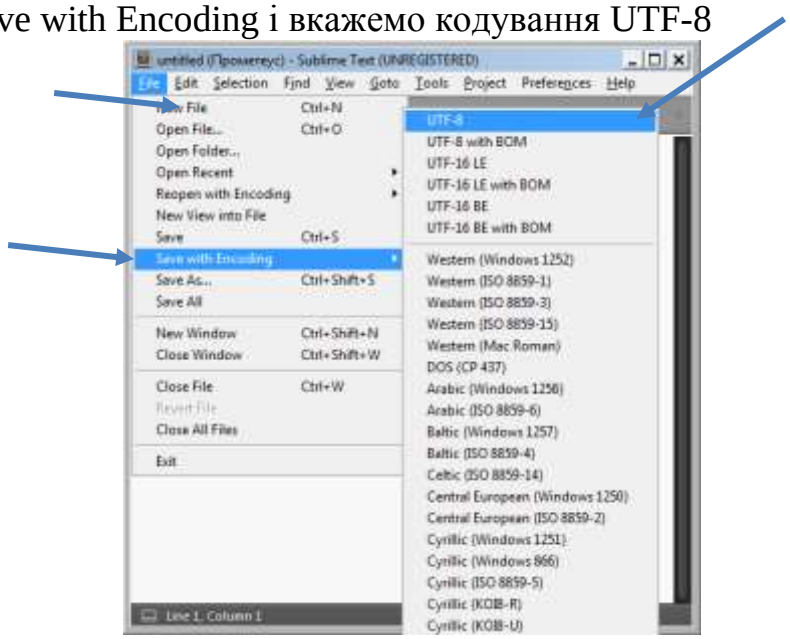
Між елементами head і body не повинно бути ніякої іншої інформації!

Робота у редакторі Sublime Text 3. Створення базової web-сторінки

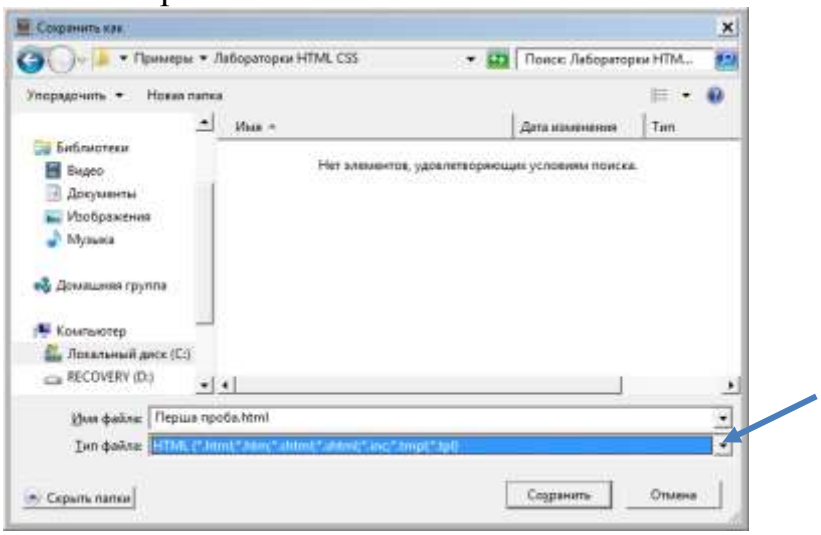
1. Відкриємо Sublime Text 3



2. В меню File оберемо New file. Потім з меню File оберемо Save with Encoding і вкажемо кодування UTF-8

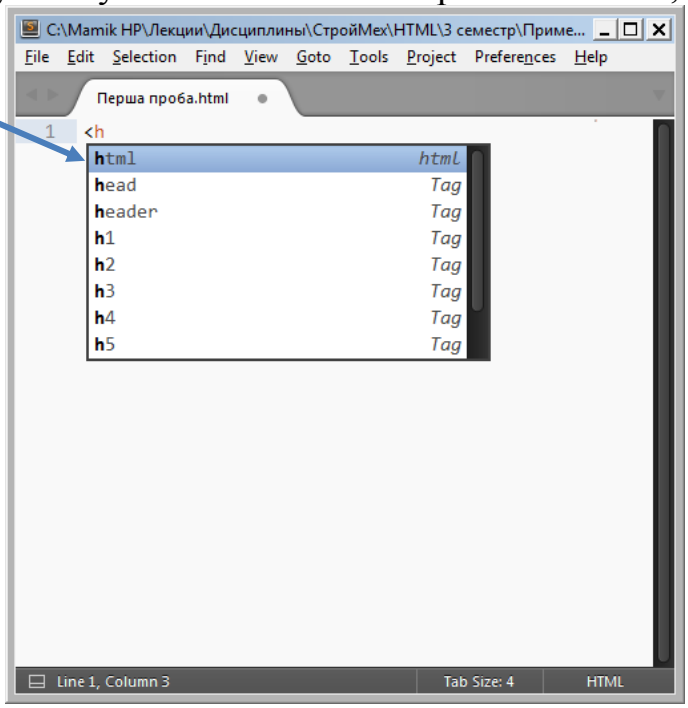


3. Дамо файлу ім’я , *наприклад* Перша проба.html
4. Із переліку типів оберемо html ….

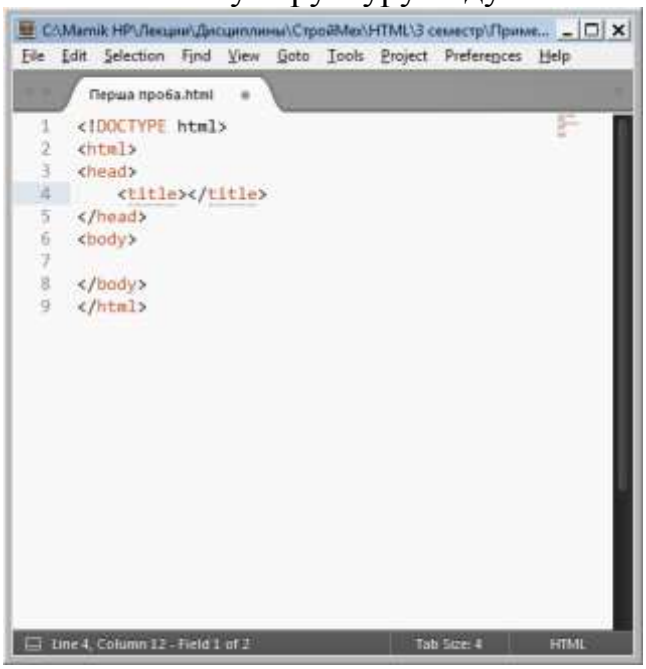


Отримаємо

Написання коду слід починати із тега <html> . Під час написання з’являється «меню-підказка», у якому кликніть мишею обраний елемент,



і Sublime Text 3 додасть всю базову структуру коду



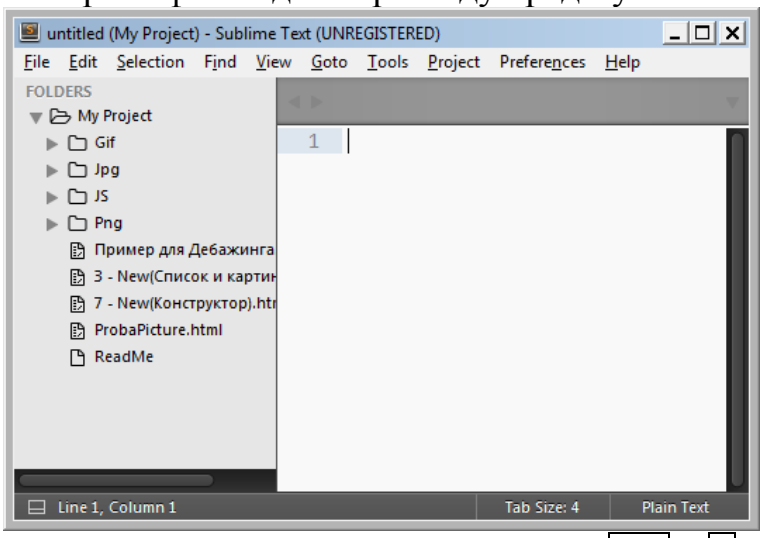
Вкажіть назву вашої сторінки і додайте у елемент head тег <meta>, що вказує на кодування, у якому збережено файл Перша проба.html:

<meta charset="utf-8">

Перед відображенням сторінки у браузері не забувайте **зберігати** внесені зміни, натискаючи комбінацію клавіш **Ctrl** + **S**.

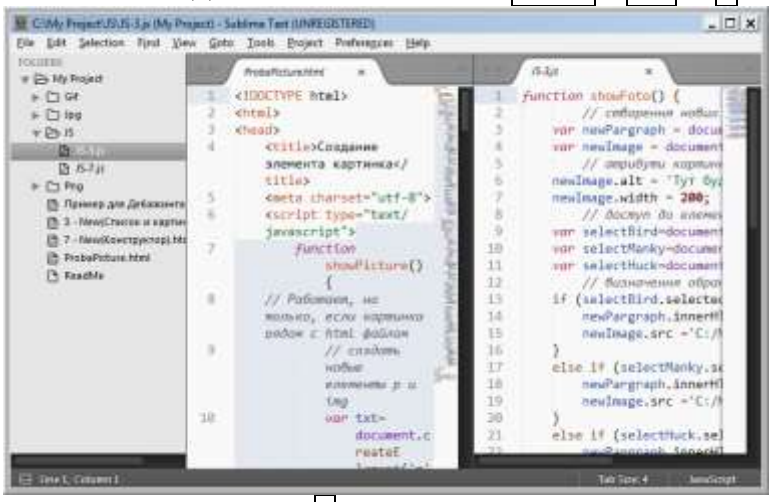
Перед тим, як зберігати файли, створіть каталог (папку) My project на диску C, наприклад, з підпапками Img, CSS, JS тощо. І файли html розміщуйте у папці My project, а файли інших типів, які будуть з'являтися поспіль вивчення у відповідний підпапці.

Наявність такої файлової структури можна використати і у редакторові Sublime для зручної роботи із проектом. Для цього в меню File редактора Sublime оберіть Open Folder і вкажіть папку My project. У редакторі з'явиться бокова панель, де можна обирати файли для перегляду і редагування вмісту:



Приховати бокову панель можна натиснувши клавіші Ctrl + B .

Для роботи (перегляду) з двома файлами можна поділити робоче вікно редактора на дві частини за допомогою клавіш Shift + Alt + 2



Повернути одну панель - натисніть 1 у попередній комбінації.

Змінити фон можна у м. Preferences ⇒ Color Scheme (дефолтний фон Monokai).

Для заміни фрагменту коду, який зустрічається кілька разів, на новий фрагмент використовуйте Ctrl + H (м. Find ⇒ Replace), а потім Ctrl + Alt + Enter .

Основні елементи сторінки

Тегова розмітка сторінки несе інформацію для обробників web- сторінок, тому надається із смислових міркувань, а не «просто за бажанням» ☺.

Виділяють дві основні категорії html елементів, які відповідають типам їх вмісту і поведінці в структурі web- сторінки – це ***блочні*** і ***строкові*** елементи. В HTML5 кожен html елемент повинен слідкувати правилам, які визначають допустимий для нього контент.

Блочні елементи – елементи вищого рівня, вони розміщуються всередині елемента <body>. Вони створюють розрив строки, утворюючи прямокутну область, і займають по ширині все вікно браузеру (або батьківський блок). Блочні елементи можуть вміщувати як строкові елементи, так і блочні елементи, але не обидва типу елементів разом.

Строкові (вбудовані) елементи не формують нові блоки контенту, а є нащадками блочних елементів. Строкові елементи вміщують тільки данні та інші строкові елементи. Винятком є тільки строковий елемент <a>, який може огортати цілі абзаци тексту, переліки, таблиці, заголовки і цілі розділи при умові, що вони не містять інших інтерактивних елементів (інших посилань і кнопок).

Розглянемо приклади тегів, які можуть міститися у елементі <body>.

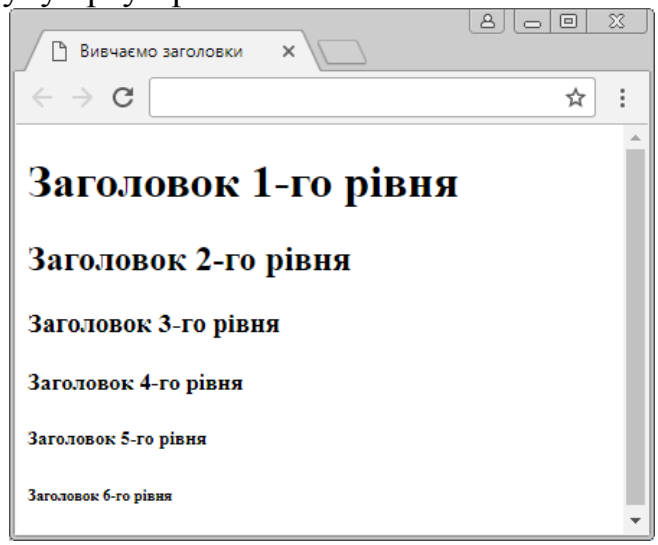
Оформлення текстів.

- **Заголовки і підзаголовки.**

Заголовки є блочними елементами і виконують важливу функцію – вказують на важливість даних, які ними огортаються. Чим вищий заголовок, тим важливіший контент. Всього заголовків шість рівнів, **h1** – найвищий, **h6** – найнижчий.
Приклад:
Створіть сторінку, додав до базової структури html-коду такий вміст елементу <body>:

```
<body>
    <h1>Заголовок 1-го рівня</h1>
    <h2>Заголовок 2-го рівня</h2>
    <h3>Заголовок 3-го рівня</h3>
    <h4>Заголовок 4-го рівня</h4>
    <h5>Заголовок 5-го рівня</h5>
    <h6>Заголовок 6-го рівня</h6>
</body>
```

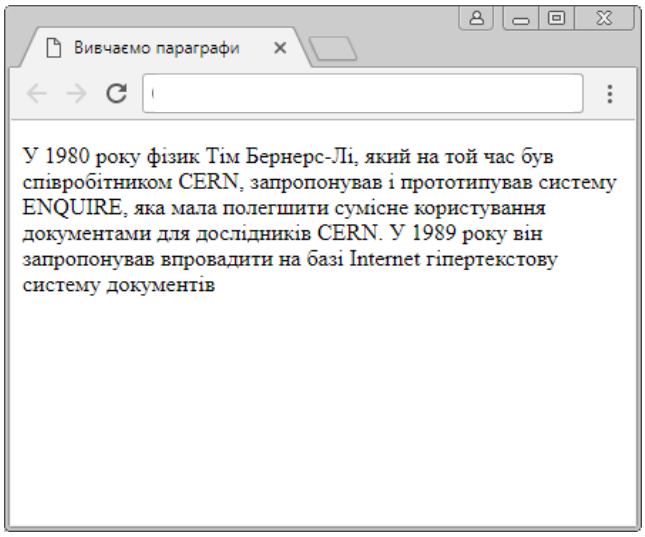
Відобразіть сторінку у браузері і ви побачите



Зазвичай при розробці сторінки використовують заголовки з першого до третього рівня включно, заголовки більш низьких рівнів практично не використовуються. Це і не дивно, бо у книгах ми теж звикли зустрічати назви розділу , у ньому міститься підрозділ, який поділений на параграфи, і ці одиниці тексту мають «типові» шрифтові виділення.
За допомогою парного строкового елемента **<small>** можна зменшити на 1 пункт висоту шрифту, який визначається блочним елементом за замовчанням.

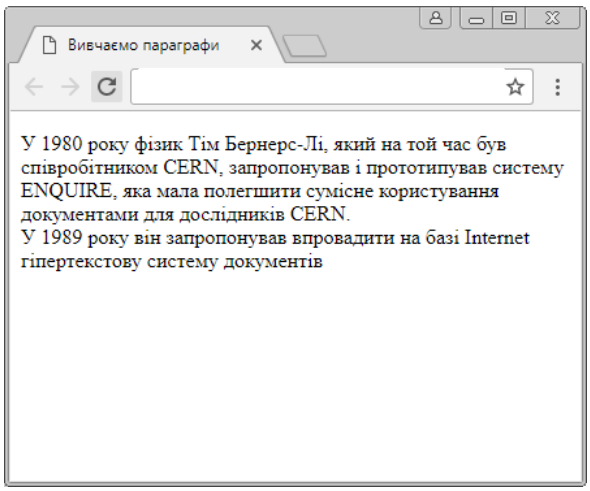
- **Текстові абзаци. Пробіли і перенос строк.**

Розподіл тексту на абзаци створюється за допомогою блочного елементу ** ** (який називають параграф). Але цей елемент є винятком серед блочних елементів і не може вміщувати всередині себе інший елемент , або інший будь-який блок.
Розглянемо *приклад*, огорнемо текст у елемент :

Вміст <body>	***У браузері***
 У 1980 року фізик Тім Бернерс-Лі, який на той час був співробітником CERN, запропонував і прототипував систему ENQUIRE, яка мала полегшити сумісне користування документами для дослідників CERN. У 1989 року він запропонував впровадити на базі Internet гіпертекстову систему документів 	

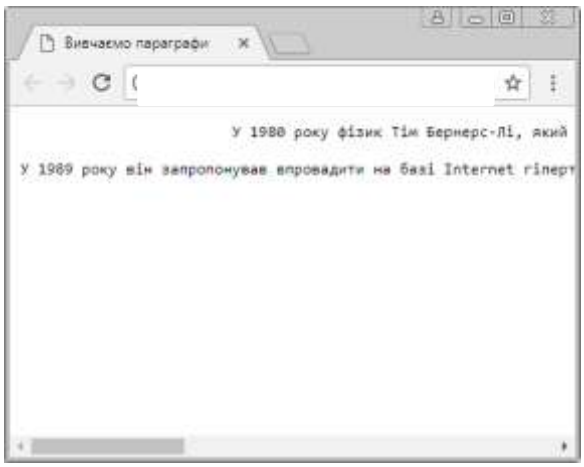
Як бачимо пропущені строки і додаткові пробіли при відображенні обробником ігноруються, і текст розташовується на всю ширину вікна браузера.

Якщо в середині параграфа ми хочемо додати розрив строки, то слід вказати строковий одинарний елемент **
** у місці «розриву».

Вміст <body>	***У браузері***
 У 1980 року фізик Тім Бернерс-Лі, який на той час був співробітником CERN, запропонував і прототипував систему ENQUIRE, яка мала полегшити сумісне користування документами для дослідників CERN. У 1989 року він запропонував впровадити на базі Internet гіпертекстову систему документів 	

Тег **<pre>** дозволяє обійти особливість елементу ігнорувати пробіли і переноси строк у абзаці. Дані, огорнуті у елемент <pre> , будуть відображені так, як ви їх набрали у тестовому редакторі, формуючи сторінку.

Вміст <body>	***У браузері***

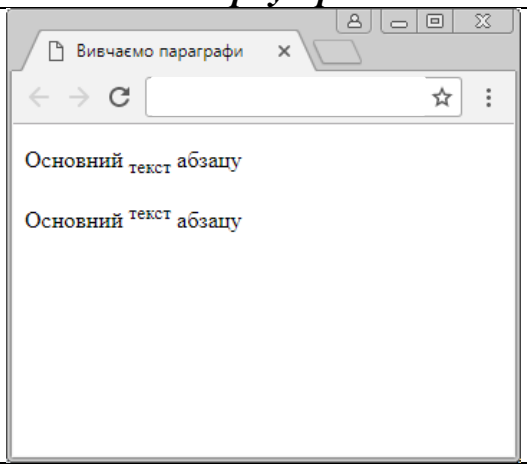
<pre>
У 1980 року фізик Тім Бернерс-Лі, який на той час був співробітником CERN, запропонував і прототипував систему ENQUIRE, яка мала полегшити сумісне користування документами для дослідників CERN.
 	
У 1989 року він запропонував впровадити на базі Internet гіпертекстову систему документів
</pre>
	

Елемент <pre> визначає блок завчасно форматованого тексту. Дані, обгорнуті цим елементом відображаються, зазвичай, моноширинним шрифтом зі всіма пробілами. Але цей елемент поступається елементу тому, що не містить автоматичного переносу слів по ширині вікна браузеру, і при використані до

абзаців тексту втрачає частину тексту. При сучасній верстці сторінок елемент <pre> практично не використовується.

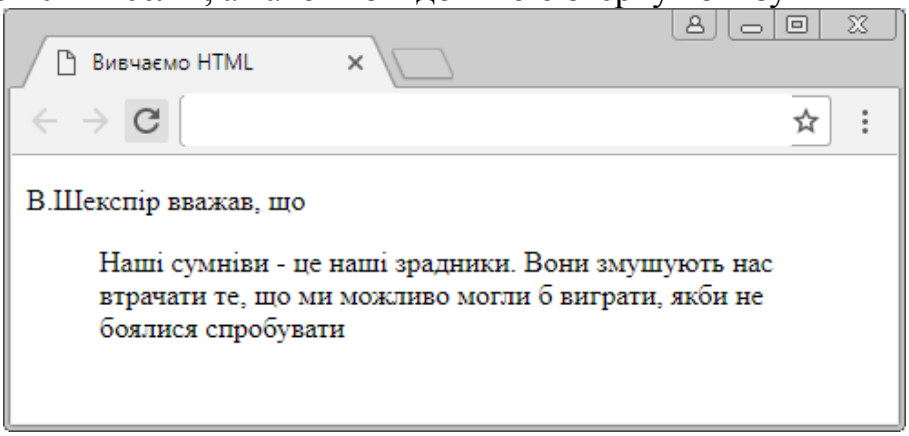
- **Надстрокові і підстрокові тексти**

Строковий парний елемент **<sub>** відображує текст, як розташований нижче базової лінії решти символів строки, і надається шрифтом зменшеного розміру. Строковий парний елемент **<sup>** відображує текст, як розташований вище базової лінії решти символів строки, і надається шрифтом зменшеного розміру

Вміст <body>	***У браузері***
 Основний _{текст} абзацу Основний ^{текст} абзацу 	

- **Цитати, переліки (списки) визначень і абревіатури**

Блочний парний елемент **<blockquote>** використовується для оформлення довгих цитат всередині документу. Текст, огорнутий цим елементом, традиційно відображується, як блок, що вирівняно із відступами зліва й справа примірно по 40 пикселів, а також із відбивкою зверху і знизу



Таку сторінку утворює наступний html-код:

```
<p>
        В.Шекспір вважав, що
</p>
<blockquote>
```

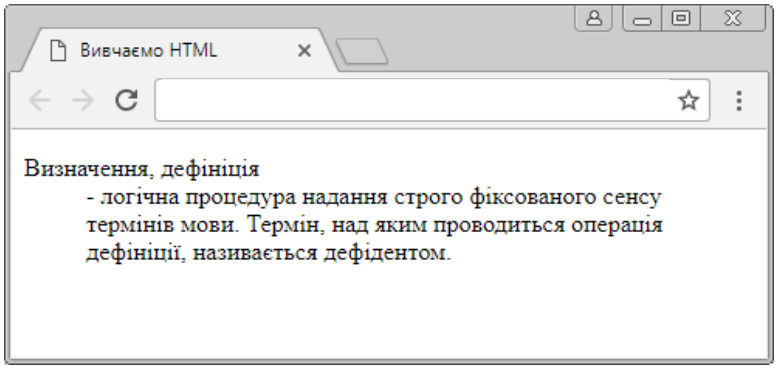
Наші сумніви - це наші зрадники. Вони змушують нас втрачати те, що ми можливо могли б виграти, якби не боялися спробувати
</blockquote>

Трійка елементів **<dl> <dt> <dd>** дають змогу створити список (перелік), якій вміщує визначення понять чи термінів, і їх відповідне розміщення на сторінці, коли термін розташовується на одній строці, а його визначення з нової строки з відступом зліва.

Наприклад,

```
<dl>
    <dt>
        Визначення, дефініція
    </dt>
    <dd>
        - логічна процедура надання строго фіксованого сенсу термінів мови. Термін, над яким проводиться операція дефініції, називається дефідентом.
    </dd>
</dl>
```

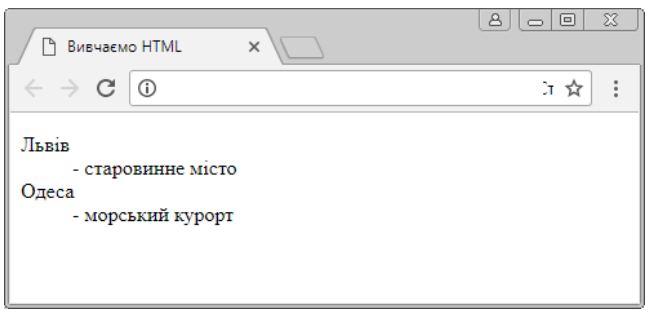
Утворює таку сторінку



Розглянемо приклад коду, що створює перелік з кількох термінів

```
<dl>
    <dt>Львів</dt>
     <dd> - старовинне місто</dd>

     <dt>Одеса</dt>
     <dd> - морський курорт</dd>
</dl>
```

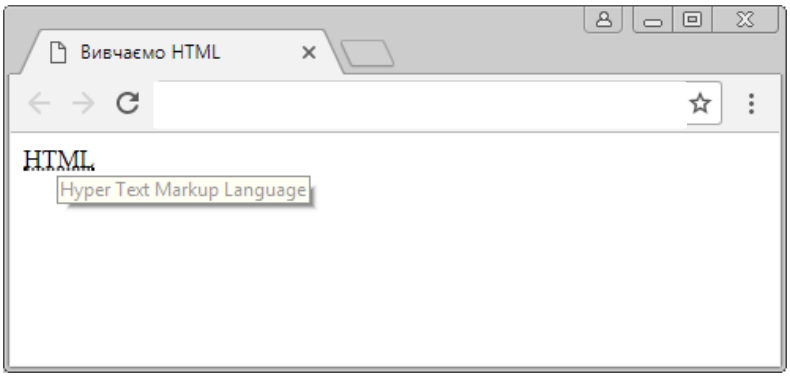
Відповідна сторінка виглядатиме так:



Елемент **<abbr>** вказує, що послідовність символів є абревіатурою. За допомогою атрибуту **title** цього елементу надається роз тлумачення скорочення, яке надає можливість зрозуміти абревіатуру тим, хто із нею не був знайомий. *Наприклад*,

```
<abbr title="Hyper Text Markup Language">HTML</abbr>
```

Вміст атрибуту title можна побачити, якщо на сторінці навести мишею на абревіатуру



Крім того, пошукові системи індексують повнотекстовий варіант скорочення, що може використовуватися для підвищення рейтингу документу.

Включення елементу <abbr> до елементу <dt> при оформленні визначення дає змогу побачити на сторінці не тільки визначення терміну, а і підказку, що «випливає», якщо навести мишею на абревіатуру, *наприклад*:

```
<dl>
    <dt>
        <abbr title="Hyper Text Markup Language">HTML</abbr>
    </dt>
    <dd>
        стандартна мова розмітки Web-сторінок в Інтернеті. Більшість Web-
        сторінок створюються за допомогою мови HTML [1] . Документ
        HTML оброблюється браузером та відтворюється на екрані у
        звичному для людини вигляді
    </dd>
</dl>
```

Марковані і нумеровані переліки (списки)

Списки дають змогу надати дані у структурованому вигляді, що доволі зручно для відображення пунктів плану дій, переліку об’єктів тощо. Для списків існують можливості оформлення за допомогою стилів CSS і обробки за допомогою JavaScript.

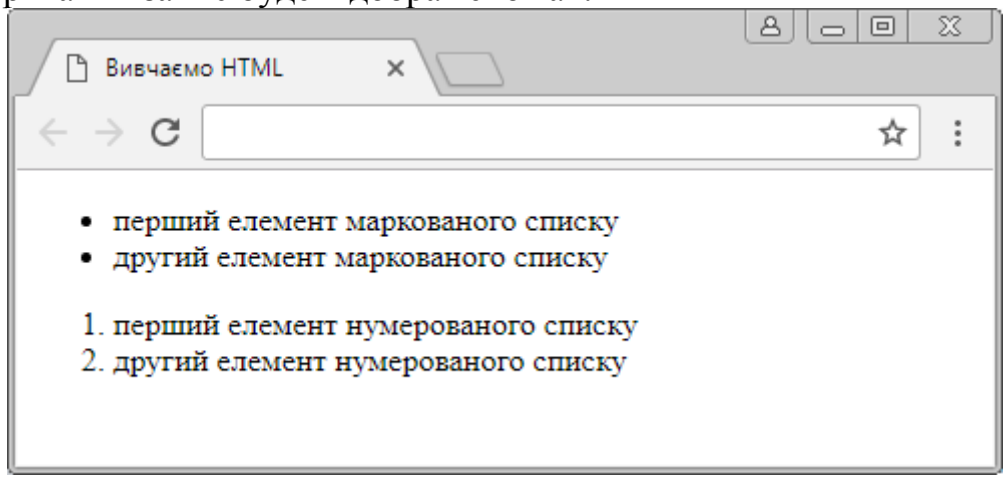
У HTML існує три вида списків:

- список визначень (його ми розглянули вище);
- список маркований;
- список нумерований

Блочні елементи **<ul>** і **<ol>** визначають яким буде список маркованим чи нумерованим, а елемент **<li>** визначатиме одиницю списку.

```
<ul>
 <li> перший елемент маркованого списку</li>
 <li> другий елемент маркованого списку</li>
</ul>

<ol>
 <li>перший елемент нумерованого списку</li>
 <li>другий елемент нумерованого списку</li>
</ol>
```

У браузері такий запис буде відображено так:



Як бачимо, перед елементами списків з'являються відповідні маркери («диск» і «цифри»). Загалом, є кілька маркерів для кожного списку:

- маркований («диск», «коло», «квадрат»);

```
<li type="disk | circle| square">
```

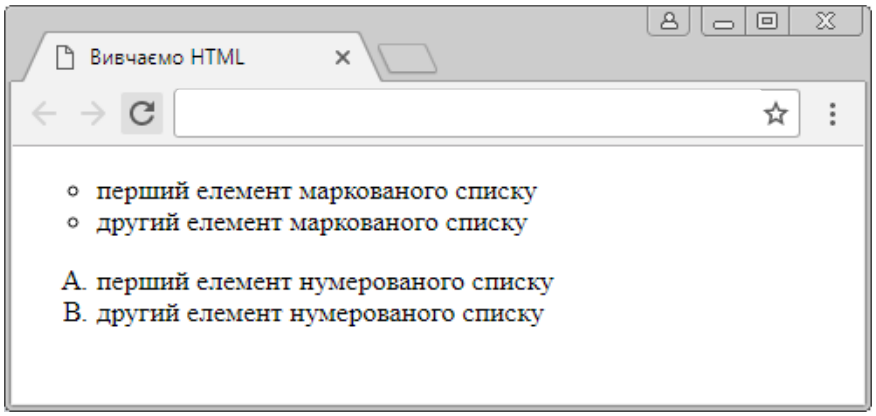
- нумерований («цифри», «великі або маленькі букви за алфавітом», «великі або маленькі букви I / i»

```
<li type="A | a| I | i | 1">
```

Для того, щоб вказати тип маркеру, слід додати атрибут **type** у елемент списку:

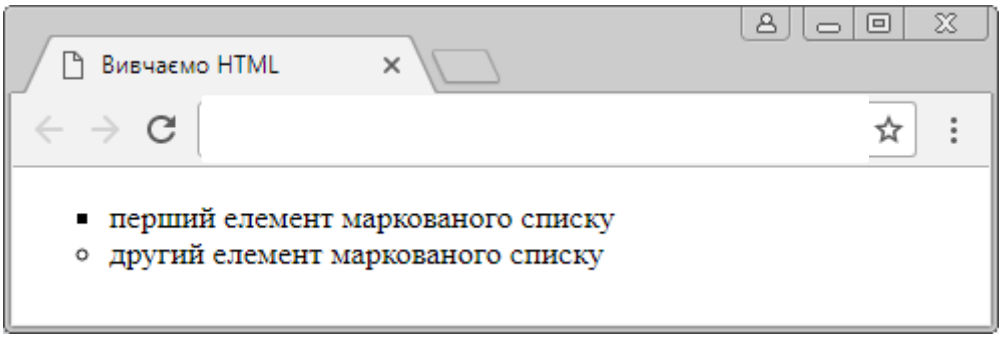
Наприклад,

```
<ul type="circle">
        <li> перший елемент маркованого списку</li>
         <li> другий елемент маркованого списку</li>
</ul>

<ol type="A">
        <li>перший елемент нумерованого списку</li>
       <li>другий елемент нумерованого списку</li>
</ol>
```



Якщо додати атрибут **type у елемент **, то вказаний у type тип маркеру буде тільки перед тим елементом списку, що містить атрибут:

```
<ul type="circle">
        <li type="square"> перший елемент маркованого списку</li>
        <li> другий елемент маркованого списку</li>
</ul>
```

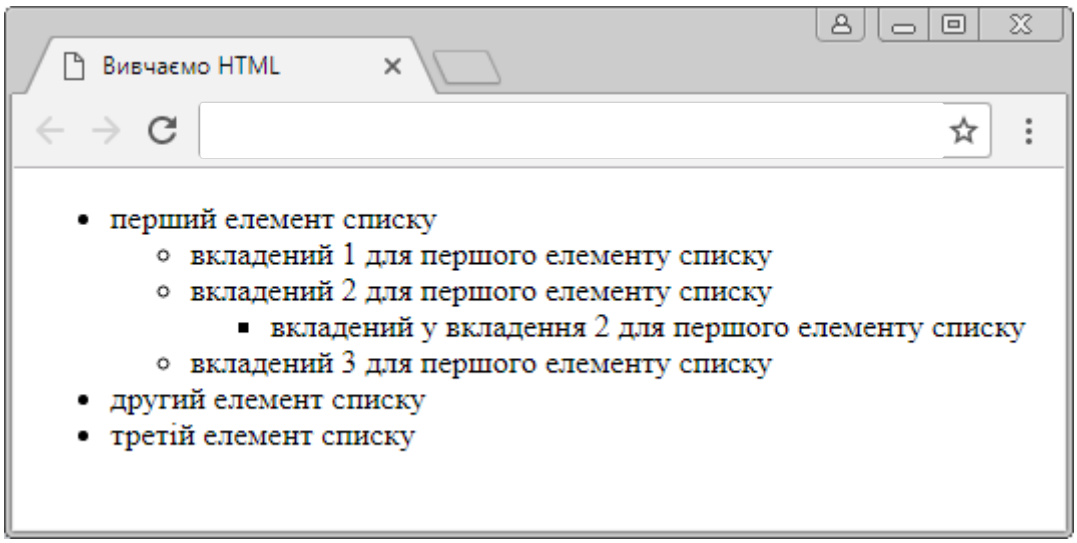


У нумерованих списках можна вказувати **початок, з якого почнеться нумерація**, за допомогою атрибуту **value**, *наприклад*, якщо нумерація починається із цифри «5», то слід записати:

```
<li type=”1” value=”5”>
```

Списки можуть бути вкладеними, типи маркерів і відступи задаються за замовчуванням, якщо не вказано інше.
Наприклад,

```
<ul>
        <li>перший елемент списку
                <ul>
                        <li>вкладений 1 для першого елементу списку</li>
                        <li>вкладений 2 для першого елементу списку
                                <ul>
                                        <li>вкладений у вкладення 2 для першого
елементу списку</li>
                                </ul>
                        </li>
                        <li>вкладений 3 для першого елементу списку</li>
                </ul>
        </li>
        <li>другий елемент списку</li>
        <li>третій елемент списку</li>
</ul>
```



Вкладати можна нумеровані списки у марковані і навпаки.

Таблиця

Таблиця складається із строк і комірок, які можуть містити текст, або картинки. Довгий час у ранніх версіях HTML таблиці із невидимою межею використовувалися для верстки web-сторінок. У HTML 5 використовують блочну семантичну верстку за допомогою спеціальних елементів, про які ми поговоримо пізніше.
Для додавання таблиці на web- сторінку використовується тег **<table>.** Цей елемент є контейнером для елементів, які визначають вміст таблиці. Будь-яка

таблиця складається із строк і комірок, які задаються відповідно за допомогою тегів **<tr>** і **<td>** .
Таблиця повинна містити хоча би одну комірку.
Є можливим замість тегу <td> використовувати тег **<th>.** Текст у комірці, оформленій за допомогою тегу <th>, буде відображений у браузері шрифтом жирного накреслення і мати вирівняння за центром комірки (інших відмінностей між <td> і <th> нема)
Загальний синтаксис для таблиці такий:

```
<table>
    <tr>
        <td> ….. </td>
    </tr>
</table>
```

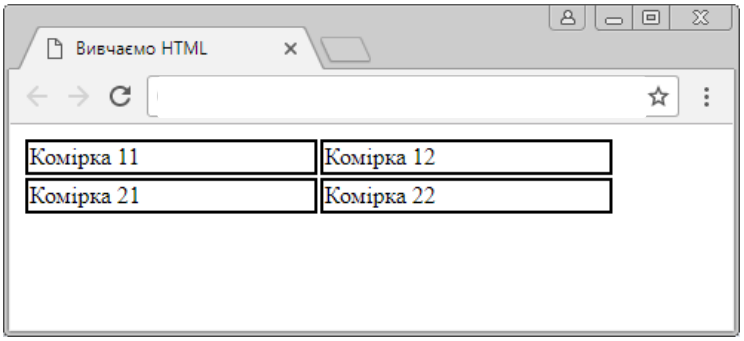
При розгляді прикладів додамо у елемент head оформлення для таблиці:
вкажімо ширину таблиці, щоб вона не займала все вікно браузера
table {width: 400px;} і мала видиму межу td {border: 2px solid black;}
Елементи оформлення слід огорнути у елемент <style>:

```
<style type="text/css">
    table {
        width: 400px;
    }
    td {
        border: 2px solid black;
    }
</style>
```

У елемент body додамо код:

```
<table>
    <tr>
        <td>Комірка 11</td>
        <td>Комірка 12</td>
    </tr>
    <tr>
        <td>Комірка 21</td>
        <td>Комірка 22</td>
    </tr>
</table>
```

Web- сторінка буде така:



При розробці коду таблиці всередині елементу table можна використовувати наступні елементи:

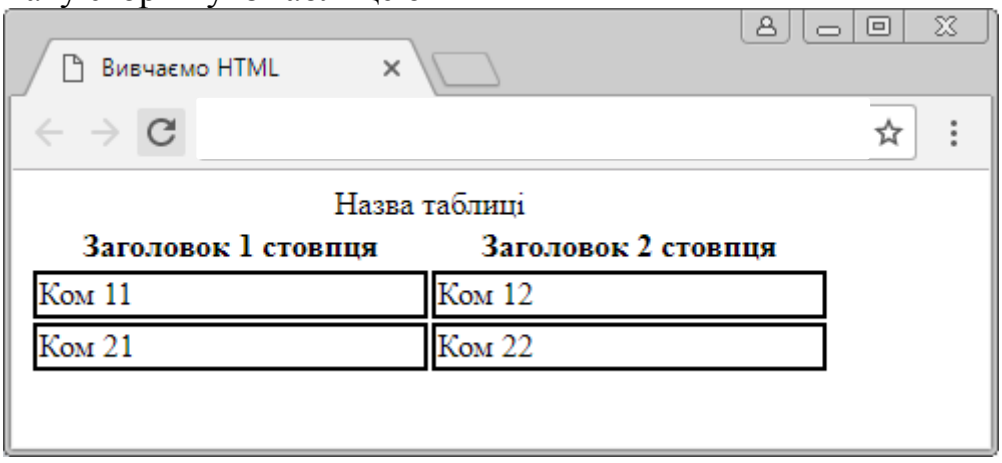
<caption> для Назви (заголовку) таблиці

<thead> , <tbody>, <tfoot> для визначення структури таблиці. Саме у елементі <thead> для строки замість елементу <tr> використовується елемент <th>.

Наприклад,

```
<table>
    <caption>Назва таблиці</caption>
    <thead>
        <tr>
            <th>Заголовок 1 стовпця</th>
            <th>Заголовок 2 стовпця</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Ком 11</td>
            <td>Ком 12</td>
        </tr>
    </tbody>
    <tfoot>
        <tr>
            <td>Ком 21</td>
            <td>Ком 22</td>
        </tr>
    </tfoot>
</table>
```

Створює таку сторінку із таблицею



Комірки у таблицях **можна об'єднувати**. Для цього використовуються атрибути:

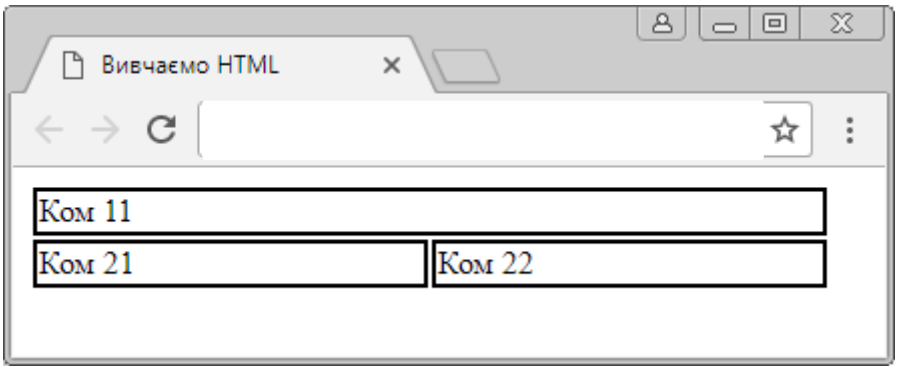
colspan для об'єднання комірок у строці

rowspan для об'єднання комірок у стовпчику.

Наприклад,

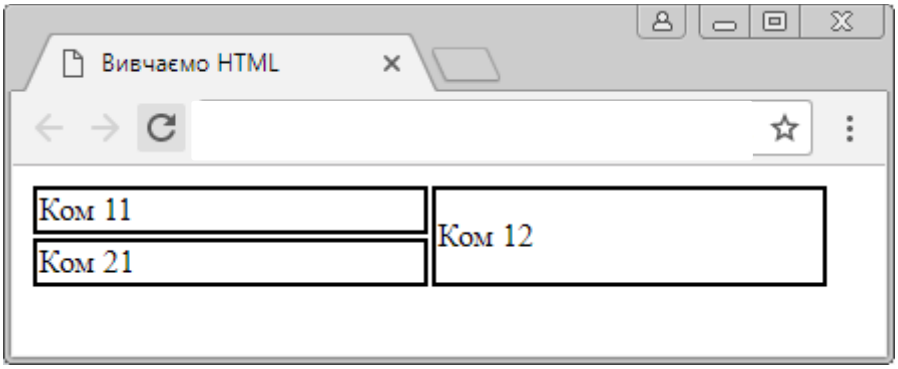
Поєднання комірок у строці:

```
<table>
    <tr>
        <td colspan="2">Ком 11</td>
    </tr>
    <tr>
        <td>Ком 21</td>
        <td>Ком 22</td>
    </tr>
</table>
```



Поєднання комірок у стовпчику:

```
<table>
    <tr>
        <td>Ком 11</td>
        <td rowspan="2">Ком 12</td>
    </tr>
    <tr>
        <td>Ком 21</td>
    </tr>
</table>
```



Рекомендація.
Якщо ви хочете створити таблицю із поєднаними комірками, то додержуйтеся при розробці такого порядку дій:

1. створіть просту («стандартну») таблицю;
2. додайте атрибут rowspan (або colspan);
3. видалите «зайві» елементи (комірки) із структури таблиці.

Посилання

Гіперпосилання - це елемент тексту на сайті, що дозволяє здійснювати перехід на інші сторінки сайту, відкривати інші документи, запускати сторонні додатки. Текст гіперпосилання на web-сторінці часто виділяється синім кольором чи підкреслюється. Гіперпосилання вважаються основним засобом навігації в середовищі Інтернет-мережі.
Розрізняють кілька видів гіперпосилань:
Зовнішні та внутрішні. Зовнішні гіперпосилання здійснюють перехід на інші сторінки або документи в мережі. Тобто на web-ресурси, які знаходяться поза сайту. Внутрішні ж гіперпосилання дозволяють нам подорожувати в межах конкретного сайту.
Якірні гіперпосилання. Використовуються для переходу в певну область документа. На web-сторінці може бути організовано зміст, в якому містяться гіперпосилання на певні частини змісту на цій же сторінці сайту. ***Якір*** – це закладка всередині сторінки, яку можна вказати у посиланні.
Оформити посилання можна за допомогою лінійного елементу <a>, загальний синтаксис має вигляд:

```
<a href = “посилання”>Текст</a>
```

У якості посилання надається:

- URL адреса для зовнішніх посилань

наприклад,

```
<a href="http://ogasa.org.ua">Академія будівництва та архітектури</a>
```

- путь доступа до файлу для внутрішніх посилань

наприклад,

```
<a href="Text/Last.html">Наступна сторінка</a>
```

- ім’я якоря для якірних посилань

наприклад,

```
<a href="#Yellow">Жовтий текст</a>
.....
<p>...</p>
<p>...</p>
<p>...</p>
```

```
<p>…</p>
…..
<p id=” Yellow”><mark>…</mark></p>
<p>…</p>
…
```

Зображення, відео, аудіо

Елемент **<img>** призначений для відображення на web- сторінці зображень у графічному форматі GIF, JPEG або PNG. Адреса файлу із картинкою задається атрибутом **src**. За допомогою атрибуту **alt** можна задати *альтернативний напис*. Альтернативний напис можна побачити на сторінці, доти картинка завантажується, або з деяких причин відсутня.

Наприклад,

```
<img src="images/figure.jpg"  alt="Вінні-Пух">
```

Елемент **<audio>** додає, відтворює і управляє настройками аудіозапису на web-сторінці. Шлях до файлу задається через атрибут **src** або вкладений тег **<source>.**

Синтаксис:

```
<audio src="URL"></audio>
```

Або так

```
<audio>
    <source src="URL">
</audio>
```

Тег **source** вставляє звуковий (або відео) файл і використовується у елементах <audio> (або <video>). Атрибутом **type** тегу source задають MIME-тип медійного джерела.

Атрибут **controls** додає панель управління до аудіофайлу.

Наприклад,

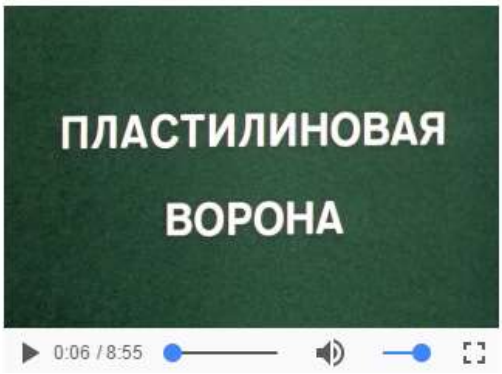
```
<audio controls>
    <source src="Audio/audПятн.mp3" type="audio/mp3">
</audio>
```

Відтворювати і управляти налаштуванням відеороликів можна за допомогою елементу <video>.
Загальний синтаксис:

```
<video>
    <source src="URL">
</video>
```

Наприклад,

```
<video controls>
    <source src="Video/Vorona.mp4" type="video/mp4">
</video>
```



Список аудіо та відеокодеків, що підтримувані браузерами, обмежений!
Для універсального відтворення у браузерах відео кодують за допомогою різних кодеків і додають ці файли одночасно.

Форми

Форми є одним з важливих елементів будь-якого сайту і призначені для обміну даними між користувачем і сервером.
Область застосування форм не обмежена відправкою даних на сервер, за допомогою скриптів користувача можна отримати доступ до будь-якого елементу форми, змінювати його і застосовувати на власний розсуд.
Документ може містити будь-яку кількість форм, але одночасно на сервер може бути відправлена тільки одна форма. З цієї причини дані форм повинні бути незалежні один від одного.
Допускається всередину контейнера **<form>** поміщати інші теги, при цьому сама форма ніяк не відображається на web-сторінці, видно тільки її елементи і результати вкладених тегів.
У якості елементів форми визначають:
Елемент **<form>** - встановлює форму на сторінці, він є обов'язковим, якщо передбачено відправляти дані з форми на сервер.
Тег **<label>** - встановлює зв'язок між певною міткою і елементом форми
Тег **<fieldset>** - призначений для групування елементів форми
Тег **<legend>** - створює заголовок групи елементів форми
«Центральним» елементом форми є **<input>** , який надає можливість створювати різні елементи інтерфейсу за допомогою атрибуту **type.**

Наведемо найбільш популярні значення атрибуту type елементу input:

- текстове поле для одно строкових текстів (**text**),
- текстове поле для багато строкових текстів (**textarea**);
- числове поле (**number**),
- надпис (**label**),
- поле з паролем (**password**) . Символи, що вводяться відображають «крапками».
- радіокнопка (перемикач) (**radio**),
- флажок (**checkbox**),
- приховане поле (**hidden**),
- кнопка (**button**),
- кнопка для відправки форми на серевер (**submit**),
- кнопка для очистки форми (**reset**),
- поле для відправки файлу (**file**)
- и кнопка із зображенням (**image**).

Для кожного елементу існує свій список атрибутів, які визначають його вид та характеристики.
Розглянемо приклади графічних елементів форми:

Одно строковий текст:

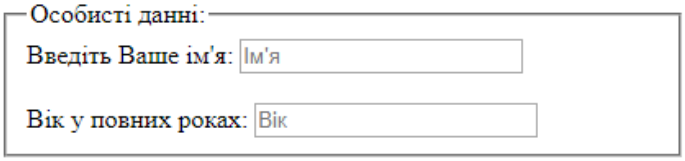
```
<input type="text" id="User_name" name="User_name" placeholder="Ваше ім'я">
```

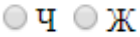
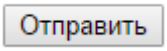
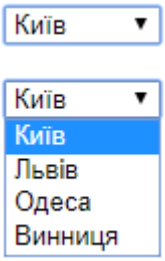
Одно строковий текст із міткою (зв'язаним надписом):

```
<label for="User_name">Ім'я:</label>
<input type="text" id="User_name" name="User_name" placeholder="Ваше ім'я">
```

Групування (fieldset) елементів форми з легендою (legend):

```
<fieldset>
        <legend>Особисті данні:</legend>
        <label for="User_name">Введіть Ваше ім'я:</label>
        <input type="text" id="User_name" name="User_name" placeholder="Ім'я" >
        <br>
        <br>
        <label for="User_age">Вік у повних роках:</label>
        <input type="number" id="User_age" name="User_age" placeholder="Вік" >
</fieldset>
```



Радіокнопка (перемикач): <input type="radio" name="Sex" value="Чол">Ч <input type="radio" name="Sex" value="Жін">Ж	
«Флажок»: <input type="checkbox" name="Remember" id="Remember">	
Кнопка «відправлення на сервер» <input type="submit" name="Send">	
Список: <select name="City"> <option value="Київ"> Київ </option> <option value="Львів"> Львів </option> <option value="Одеса">Одеса</option> <option value="Винниця"> Винниця </option> </select>	
Повідомлення: <textarea name="message" id="message">Your message</textarea>	

Примітка. Зверніть увагу на наявність атрибуту name="Sex" у елементі <input type="radio">, наявність цього атрибуту з однаковим ім'ям для всіх input забезпечує обрання тільки однієї радіокнопки.

Завдання для самостійної роботи

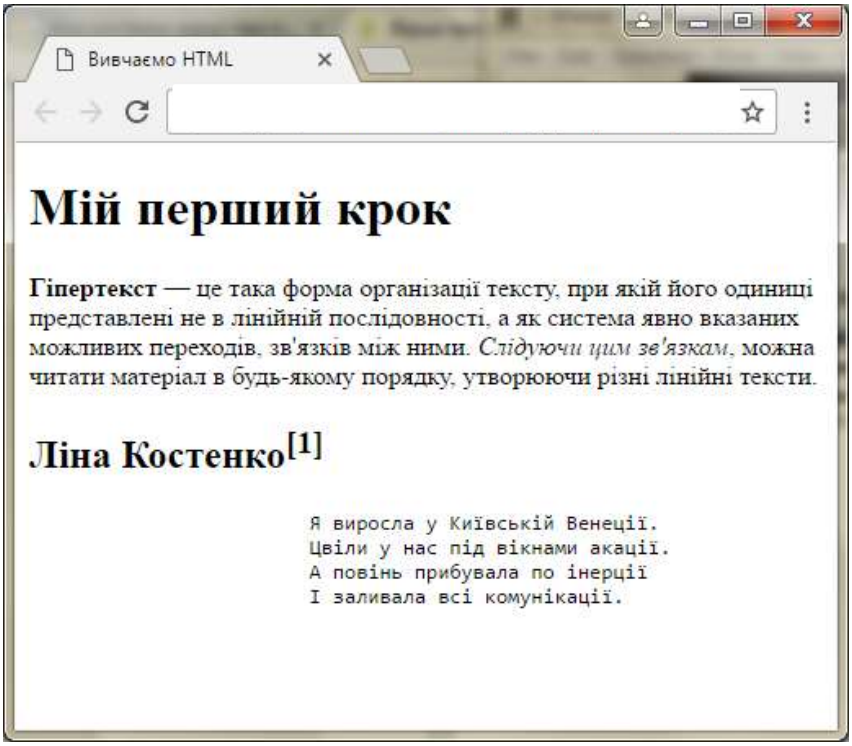
Завдання 1.

Розробить html-код сторінки з «титулом» Вивчаємо HTML (як на мал.3.1). Додайте на сторінку:

- заголовок (h1) ;
- параграф із тестом;
- підзаголовок (h2) ;
- текст із збереженням «пунктуації»

Зверніть увагу на текстові фрагменти, що містяться на сторінці. Тут:

- слово «**Гіпертекст**» огорнуто в елемент
- фраза «*Слідуючи цим зв'язкам*» огорнута в елемент
- символи «[1]» є надстроковими

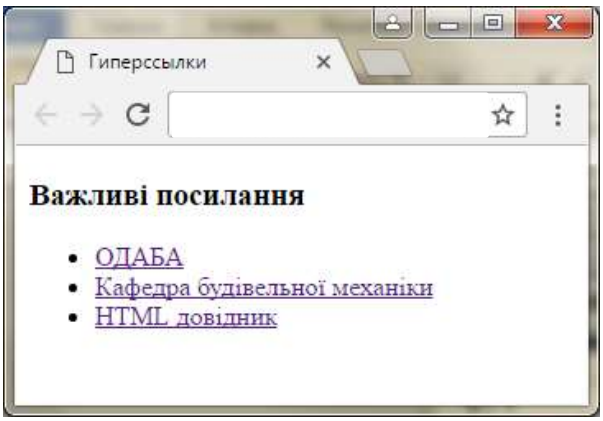


Мал. 3.1.

Завдання 2.

Створіть маркований список, елементами якого є URL посилання (див. Мал.3.2.)

http://www.ogasa.org.ua
http://budmeh.od.ua
http://htmlbook.ru/html

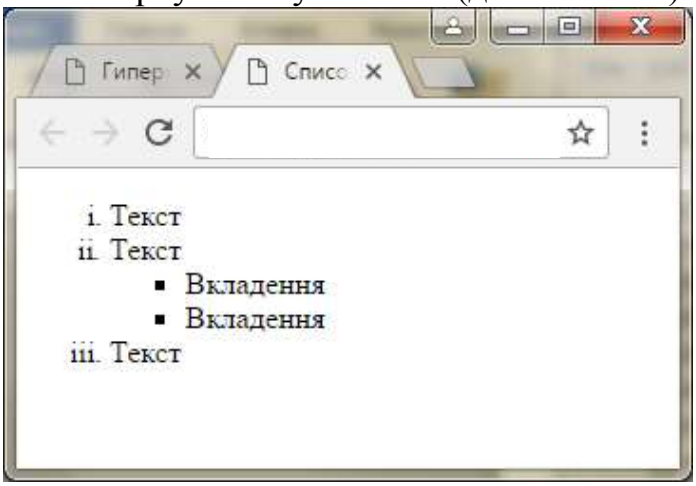


Мал. 3.2.

Завдання 3.

Створить маркований список із двох елементів, який вкладено у другий елемент нумерованого списку із трьох елементів.

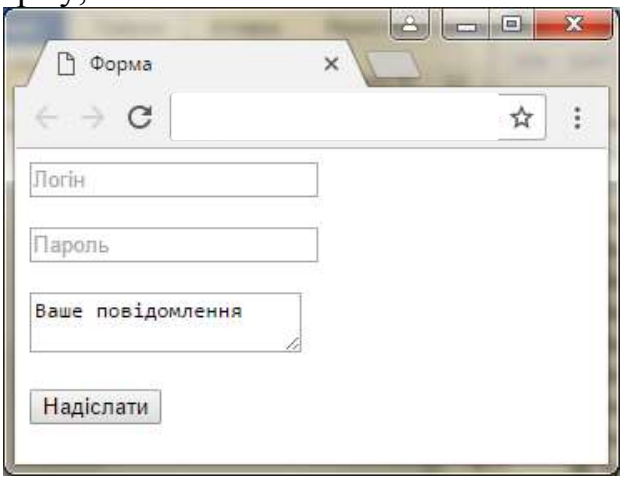
Зверніть увагу на типи маркування у списках (див. Мал.3.3)



Мал. 3.3

Завдання 4.

Створить простую форму, як на Мал. 3.4.



Мал.3.4.

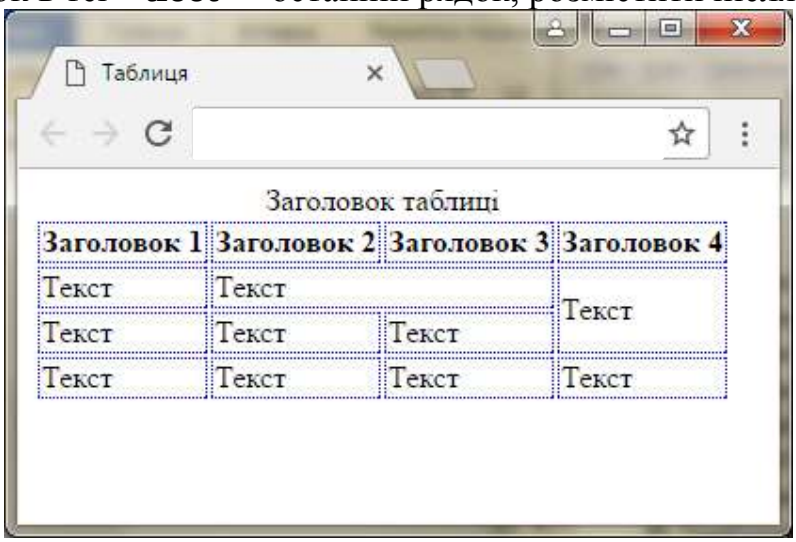
Завдання 5.

Створіть структуровану таблицю, яка містить об’єднані комірки (див. Мал.3.5.) Для того, щоб таблиця мала видиму межу, додайте у розділ <head> .

```
<style type="text/css">
        th,td {
                border: 1px dotted blue;
        }
</style>
```

Вимоги:

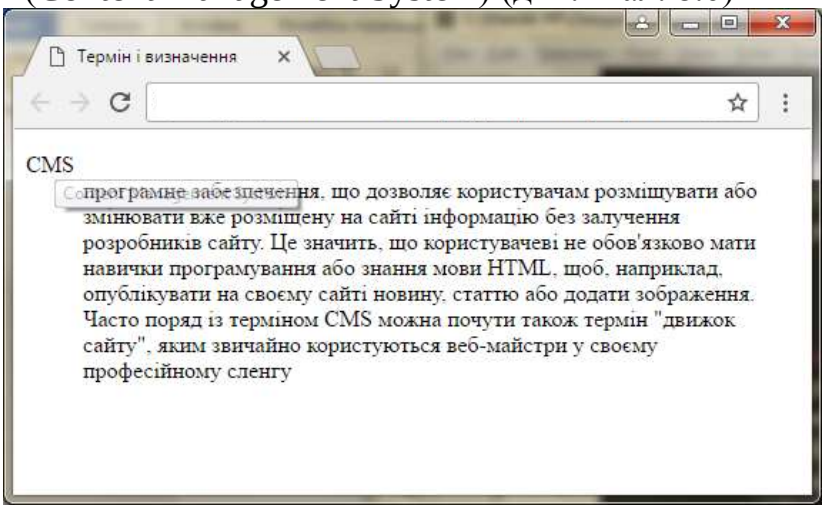
- Заголовок в тег **<caption>**
- Заголовки колонок в тег **<thead>**
- Основна інформація в тег **<tbody>** - два рядки
- Висновок в тег **<tfoot>** - останній рядок, розмістити після </tbody>



Мал.3.5.

Завдання 6.

Створить текст, як термін і визначення. Додайте до абревіатури CMS «підказку» (Content Management System) (див. Мал. 3.6)



Мал.3.6.

Семантична верстка web-сторінок

Семантика (фр. Sémantique від грец. Σημαντικός - визначає) - наука про розуміння певних знаків, послідовностей символів і інших умовних позначень.
HTML - це теж мова, його «слова» - теги, мають певний логічний зміст і призначення. Тому, в першу чергу, семантичний html-код - це верстка з правильним використанням тегів, використанням їх за призначенням, так як їх задумували розробники мови HTML і Web стандартів.
Розмітка сторінок з урахуванням застосування тегів за їх призначенням, отримало назву POSH (Plain Old Semantic HTML). Термін «народився» у 2007 році.
Підхід до створення web - сторінки на підставі семантичної верстки протиставляється підходу, при якому написання html-коду визначається зовнішнім виглядом web –сторінки. Стандарт HTML з самого початку містив у собі ряд семантичних тегів, але більшу популярність семантична верстка отримала після початку робіт над HTML5.
Семантична верстка передбачає логічну і послідовну ієрархію для побудови всієї web - сторінки, відповідно до законів HTML-документа. Семантична верстка і розмітка web- сторінки видна браузеру і роботам. Семантична верстка і розмітка дозволяє більш точно визначати значимість окремих елементів web-сторінки і всього тексту в цілому Тому, перш за все - семантична верстка потрібна для поліпшення робото-функціоналу сайту і, як наслідок - кращої його пошукової індексації.

Спеціальні елементи div і span

Лінійний контейнер **** призначений для виділення частини інформації всередині інших елементів, з метою подальшого встановлення для цієї інформації оформлення за допомогою каскадної таблиці стилів CSS. Тег часто використовується з атрибутами **id** або **class** і іменем відповідного селектору.

```
<p>Ділянка тексту <span style="color:#ff0000">червоного</span> кольору.</p>
```

В браузері побачимо

Ділянка тексту червоного кольору.

Зазвичай span міститься у елементах <p> або , але не навпаки, і не має власних дефолтних властивостей.

«Базовий контейнер» , елемент ** ** є блоковим елементом і призначений для виділення фрагмента документа з метою зміни виду вмісту. Як правило, вид блоку управляється за допомогою стилів CSS. Щоб не описувати кожен раз стиль всередині тега, можна виділити стиль в зовнішню таблицю стилів, а для тега додати атрибут **class** або **id** з ім'ям селектора.

Як і при використанні інших блочних елементів, вміст тега завжди починається з нового рядка, але інших дефолтних властивостей не має.
Елемент div із відповідним селектором у ранніх версіях HTML зазвичай використовувався, як елемент семантичної верстки.

Семантичні тегі HTML5

У HTML5 для структури коду введено кілька нових тегів: <article>, <aside>, <footer>, <header>, <nav> та ін., які замінюють в деяких випадках звичний . Хоча здається, що особливої різниці між тегом із селектором *наприклад,*) і <header> немає, між ними лежить величезна прірва. Теги орієнтовані не на людей, яким немає сенсу заглядати в вихідний код сторінки, а на машини, що інтерпретують код. Машини або роботи не розуміють , для них це типовий тег розмітки - заміни його на і сенс не зміниться. Інша справа елемент <header>, робот, виявивши цей тег, сприймає його саме як «шапку» сайту або розділу.
В стилях для нових блоків доцільно ставити display: block.

Надамо приклади і пояснимо область застосування семантичних тегів.

<header> - визначає «шапку» сайту чи розділу. Зазвичай містить логотип, назву компанії тощо.

<footer> - задає «підвал» сайту чи розділу, містить ім'я розробника документу, дату створення, контактну і правову інформацію.

<article> - задає зміст документу, який містить новини, статті, записи блогу, форуму тощо

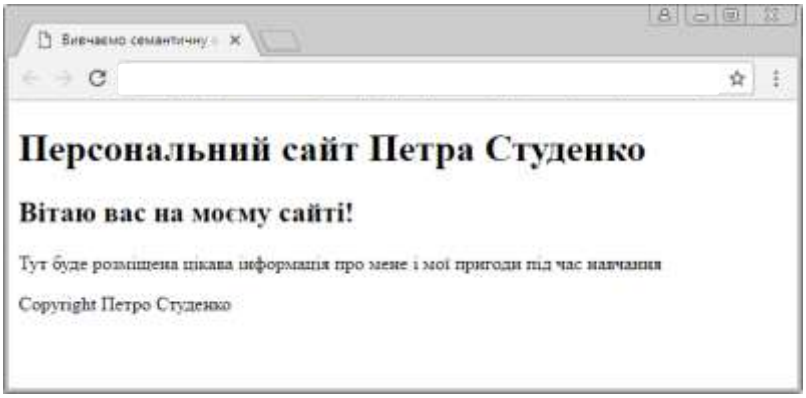
<main> - призначений для основного вмісту документа. Вміст має бути унікальним і не включати типові блоки на зразок «шапки» сайту, «підвалу», навігації, бокової панелі, форми пошуку тощо. Аналогічний тегу <article>, але, на відміну від тегу <article>, яких може бути на сторінці кілька, тег <main> повинен бути на сторінці один (він вказує на унікальний вміст).

Наприклад,
Наступний html-код

```
<!DOCTYPE html>
<html>
    <head>
        <title>Вивчаємо  семантичну верстку HTML</title>
        <meta charset="utf-8">
    </head>
```

```
        <body>
            <header>
            <h1>Персональний сайт Петра Студенко</h1>
            </header>
            <article>
              <h2>Вітаю вас на моєму сайті!</h2>
              <p>Тут буде розміщена цікава інформація про мене і мої
    пригоди під час навчання</p>
            </article>
            <footer>
            Copyright Петро Студенко
            </footer>
        </body>
    </html>
```

У браузері виглядає так



Як бачимо, наявність семантичних тегів ніяк не позначилася на вигляді сторінки, але для обробників web- сторінок наявність семантики є найважливою.

<aside> - визначає блок, який не належить до основного контенту. Використовується для розміщення рубрик, посилань на архів, міток та іншої інформації. Такий блок, якщо він розташовується збоку, називається, як правило, «сайдбар» або «бокова панель».

<nav> - задає навігацію по сайту. Якщо на сторінці кілька блоків посилань, то в <nav> зазвичай поміщають пріоритетні посилання. Також допустимо використовувати кілька тегів <nav> в документі. Забороняється вкладати <nav> всередину <address>.

Наприклад,

```
<header>
    <h1>Чебурашка і крокодил Гена</h1>
 </header>
```

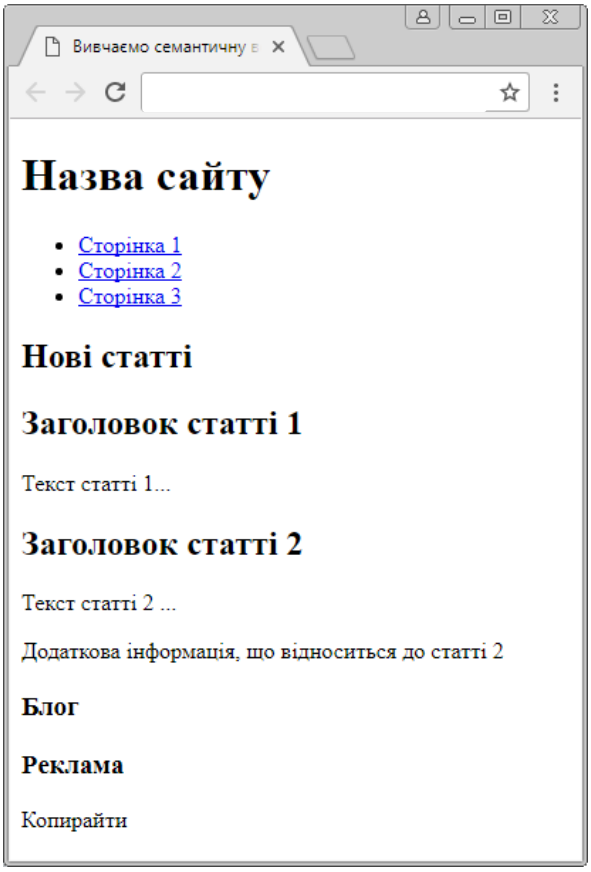
```
<nav>
      <a href="1.html">Чебурашка</a> | <a href="2.html">Гена</a> |
      <a href="3.html">Шапокляк</a> | <a href="4.html">Лариска</a>
</nav>
<article>
   <h2>Ласкаво просимо!</h2>
</article>
```

<section> - визначає розділ документа, який може включати в себе заголовки, «шапку», «підвал» і текст. Допускається вкладати один тег <section> всередину іншого.
Об'єкти, які розташовуються усередині <section>, об'єднані загальним змістом. Цей елемент також використовують для розбиття тексту на розділи.

Наприклад,

```
<body>
      <header>
            <h1>Назва сайту</h1>
            <nav>
                  <ul>
                    <li><a href="page1.html">Сторінка 1</a></li>
                    <li><a href="page2.html">Сторінка 2</a></li>
                    <li><a href="page3.html">Сторінка 3</a></li>
                  </ul>
            </nav>
      </header>
      <section>
              <h2>Нові статті</h2>
              <article>
                  <h2>Заголовок статті 1</h2>
                  <p>Текст статті 1...</p>
              </article>
              <article>
                  <h2>Заголовок статті 2</h2>
                  <p>Текст статті 2 ...</p>
                 <aside>
                     Додаткова інформація, що відноситься до статті 2
                 </aside>
              </article>
      </section>
      <aside>
```

```
            <section>
                <h3>Блог</h3>
            </section>
            <section>
                <h3>Реклама</h3>
            </section>
    </aside>
    <footer>
      <p>Копирайти</p>
    </footer> </body>
```



<hgroup> - використовується для групування заголовків web- сторінки або розділу.

<figure> - використовується для групування будь-яких елементів, *наприклад*, зображень і підписів до цих зображень.

<figcaption> - містить опис для тегу <figure>. Тег <figcaption> повинен бути першим або останнім елементом у групі.

Валідація коду

Валідацією будемо називати перевірку документа на відповідність web-стандартам і виявлення існуючих помилок. Відповідно, *валідним* є такий web-документ, який пройшов подібну процедуру і не має зауважень до коду. Код web- сторінки повинен підкорятися певним правилам, які називаються специфікацією, її розробляє W3 Консорціум (www.w3c.org) за підтримки розробників браузерів.
Перевірити код на валідність можна на сайті https://validator.w3.org/ , де у наданому полі слід ввести адресу до коду, що перевіряється. Після перевірки ви побачите можливі помилки чи повідомлення про успішне проходження валідації.

Спеціальні символи в HTML

Для відображення символів, що відсутні на клавіатурі, використовуються спеціальні знаки, які починаються із амперсанду (**&**) і закінчуються крапкою із комою (**;**).

В таблиці наведені деякі найбільш популярні спец знаки:

Опис символу	***Знак, що буде відображений***	***Код html***
знак авторського права	©	©
охоронний знак	®	®
параграф	§	§
плюс-мінус	±	±
амперсанд	&	&
знак «менше»	<	<
знак «не більше»	≤	≤
знак «більше»	>	>
знак «не менше»	≥	≥
знак «нерівності»	≠	≠
знак «наближеного рівняння»	≈	≈
буква мю (грецька строкова)	μ	μ
буква епсілон (грецька строкова)	ε	ε
буква дельта (грецька строкова)	δ	&delta
буква дельта (грецька прописна)	Δ	Δ
конверт	✉	✉
смайлик	☺	☺
знак телефону	✆	✆

Додаткові символи можна подивитися тут
http://htmlbook.ru/samhtml/tekst/spetssimvoly
І тут https://html5book.ru/specsimvoly-html/

Додавання іконки сайту у адресний рядок сайту

Під час завантаження сайту у адресному рядку, а потім і у заголовці сайту можна відобразити невеличку картинку – ***іконку сайту***, яка пов'язана із тематикою сайту.

Щоб створити таку «фічу» ☺ треба за допомогою будь-якого графічного редактора, наприклад Photoshop, намалювати зображення розміром 16×16 пікселів, дати зображенню ім'я favicon.ico і розташувати цей файл у кореневому каталозі сайту. Для більшості браузерів цього буде досить, щоб відобразити малюнок під час завантаження сайту у його заголовці. В Інтернеті існують і інші програми для створення іконок, наприклад http://favicon.ru/ , а для конвертації файлів до типу ico конвертери файлів, наприклад, https://image.online-convert.com/ru/convert-to-ico.

У якості іконки можна також використовувати файли типів png і gif, але для їх додавання у тегі <head> слід записати, *наприклад*:

```
<link rel="shortcut icon" href="images/favicon.ico" type="image/x-icon">
```

У атрибуті href записуємо шлях до файлу favicon.ico;
а атрибут type змінюємо на image/gif або image/png, в залежності від типу зображення.

Основи CSS

Спочатку мова розмітки HTML передбачала тільки тегі розмітки для групування тексту, але згодом стала поширюватися тегами оформлення, *наприклад*, такими, як <font>, що призводило до перемішування розмітки з оформленням, і робило коди нечитабельними.

Вирішити цю проблему намоглися наприкінці 1996 року, коли Консорціум Всесвітнього Павутиння W3C надав стандарт CSS.

CSS - Cascading **S**tyle **S**heets (Каскадні документи стилів) дозволяє зберігати інформацію про оформлення HMTL документа в окремому зовнішньому файлі з розширенням .css. Редагуючи лише один цей файл стало можливим зміна оформлення цілого веб-сайту.

На даний момент CSS є стандартом оформлення HTML документів та підтримується всіма сучасними браузерами. Версія, яку ми будемо розглядати, CSS3.

Модель візуального форматування CSS являє собою алгоритм, який обробляє HTML-документ і виводить його на екран пристрою. Дана модель перетворює кожен елемент документа таким чином, що він генерує нуль або більше прямокутних боксів відповідно до блокової моделі CSS. Положення цих боксів на сторінці визначається наступними факторами:

- розмірами елемента (з урахуванням того, задані вони явно чи ні); типом елемента (строковий або блочний);
- схемою позиціонування (нормальний потік, позиційований або плаваючі елементи);
- відносинами між елементами в DOM;
- внутрішніми розмірами зображень, які містяться на сторінці;
- зовнішньою інформацією (*наприклад*, розміри вікна браузера).

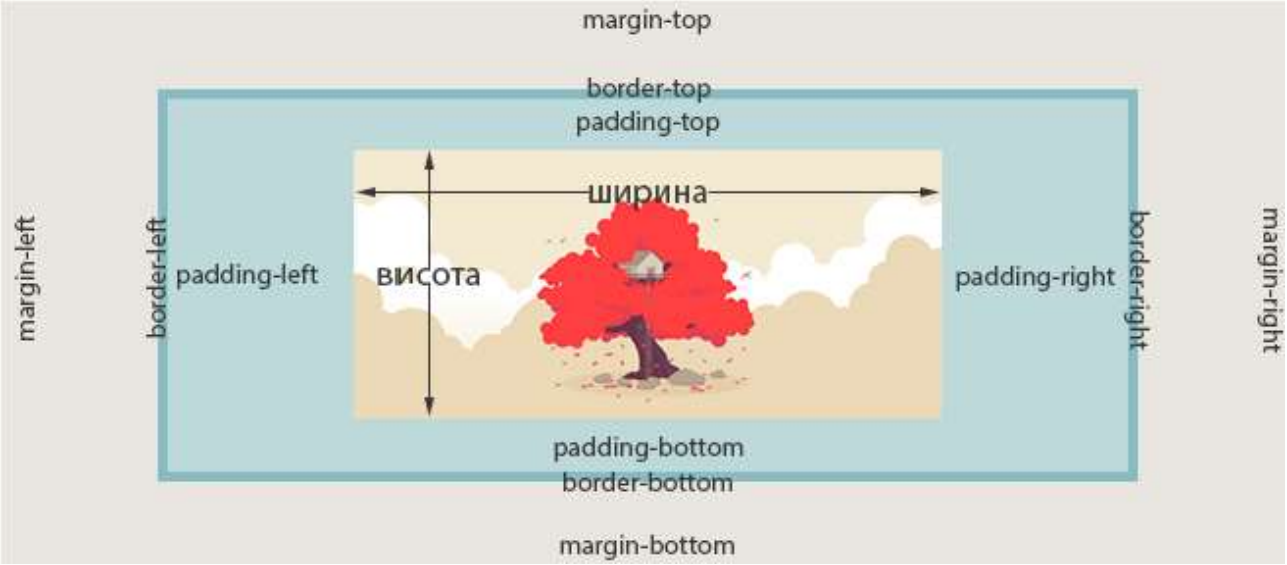
Положення і розміри боксу кожного елемента розраховуються щодо країв прямокутної області блоку, яка містить бокс, так званого «***containing block***». Розміри боксу не обмежуються розмірами containing block, тому за певних умов він може виходити за його межі.

Блочні елементи можуть розміщуватися безпосередньо всередині елемента <body>. Вони створюють розрив рядка перед елементом і після нього (так звану *відбивку*), утворюючи прямокутну область, яка по ширині займає всю ширину web- сторінки або батьківського блоку (якщо для елемента не задано значення width).

Строкові елементи є нащадками блочних елементів. Вони ігнорують верхні і нижні margin і padding, але якщо для елемента задано фон, він буде поширюватися на верхній і нижній padding, заходячи на сусідні рядки тексту.

Ширина і висота строкового елемента залежить тільки від його вмісту, задати розміри за допомогою CSS не можна. Але можна збільшити відстань між сусідніми елементами по горизонталі за допомогою горизонтальних полів і відступів.

Блочна модель елементу має вигляд



Область вмісту - це вміст елемента, *наприклад*, текст або зображення.

Внутрішній відступ - це відстань між основним вмістом і його межею (задається властивістю **padding**). Якщо для елемента задати фон, то він поширюється також і на поля елемента. Внутрішній відступ не може приймати від'ємних значень, на відміну від зовнішнього відступу.

Зовнішній відступ додає відстань ззовні елемента від зовнішньої рамки боксу до сусідніх елементів, тим самим розділяючи елементи на сторінці (задається властивістю **margin**). Зовнішні відступи завжди залишаються прозорими і через них видно фон батьківського елемента.

Значення **padding і margin** задаються в наступному порядку: **верхнє, праве, нижнє і ліве.**

Межа, або рамка елемента, задається за допомогою властивості **border**. Якщо колір рамки не заданий, вона приймає колір основного вмісту елемента, *наприклад*, тексту. Якщо рамка має розриви, то крізь них буде проступати фон елемента.

Зовнішні, внутрішні відступи і рамка елемента не є обов'язковими, за замовчуванням їх значення дорівнює нулю. Тим не менш, деякі браузери додають цим властивостям позитивні значення за замовчуванням на основі своїх таблиць стилів.

Одиниці вимірювання

Вимірювати розміри елементів можна за допомогою статичних або відносних одиниць. Десяткові числа записують із використання розділителя крапка «**.**». Назву одиниці вимірювання до числа додають без пробілу.

Статичні одиниці. Є зручними при вказанні розмірів при друці на паперовий носій*:*

mm - міліметри, міжнародна система виміру;

cm – сантиметри, міжнародна система виміру;

in – дюйми, Британська система виміру, дорівнюють 2.539см;

pt – стандартна типографська одиниця виміру, зазвичай використовується для указання кеглю (розміру шрифту), дорівнює 1/72,27in;

Відносні одиниці:

% - відсотки (1/100 частина). У кожному конкретному випадку 1% може мати інший розмір. Одиниця виміру зручна для надання ширини і висоти блоків;

px – пікселі, 1px відповідає одній крапці на моніторі. Пікселі зручні для суміщення кількох графічних зображень;

em – кегельна шпація, 1em відповідає висоті рядка прописних (великих) букв, тобто її розмір змінюється в залежності від кегля шрифта. При крупному кеглі вона більша, при дрібному – менша. Використання кегельної шпації для розмірів гарантує, що відношення між розмірами елементів на сторінці збережеться при будь-якому кеглі шрифту;

ex – висота строчних (маленьких) букв, аналогічна кегельній шпації, але відраховується від висоти строчних букв конкретного шрифту.

Старайтеся використовувати відносні одиниці виміру, такий підхід робить web-сторінку більш гнучкою до розміру екрану пристрою, на якому її відображують.

Правила написання CSS

CSS, як і будь-яка мова, має свій синтаксис. У ньому немає ні елементів, ні параметрів, ні тегів. У ньому є *правила*. Правило складається з селектора і блоку оголошення стилів, укладеного у фігурні дужки:

СЕЛЕКТОР {БЛОК ОГОЛОШЕННЯ СТИЛІВ}

Оголошення стилю записується так:

ВЛАСТИВІСТЬ : ЗНАЧЕННЯ ;

Наприкінці кожного оголошення ставиться крапка із комою «**;**».
Наприклад,

h1 {font-size: 30px;}

селектор властивість значення

селектор – це певне ім'я стиля, зв'язане із елементом з html-коду, до якого застосовується стиль. У якості селекторів виступають назви тегів, ідентифікаторів, класів, атрибутів тощо.
Кілька оголошень для конкретного селектора можна записати так, *наприклад*:

```
h1 {font-size: 30px; color: red;}
```

Для того, щоб CSS був читабельним корисно додержуватися таких правил:

- ✓ кожний селектор пишіть на окремому рядку;
- ✓ кожну властивість селектору з блоку оголошень стилів пишіть на окремому рядку;

```
h1 {
    font-size: 30px;
    color: red;
}
```

- ✓ властивості, які оказують на документ більш впливову дію, пишіть першими. Загальна логіка написання така «від глобального – до локального і менш важливого», а саме:
  1. спочатку положення елементу відносно інших елементів;
  2. потім розміри і відступи (зовнішні і внутрішні) елементу;
  3. якщо потрібно, рамку (border). Вона теж відноситься до розмірів;
  4. загальне оформлення вмісту (list-style-type, overflow…);
  5. наприкінці додають колір і стилеве оформлення (background, color, font…);

До стилю можна додавати коментарі. Синтаксис коментаря такий

/* тут коментар ***/**

Коментар може міститися як після правила на одному з ним рядку, так і окремим рядком.

Скорочений запис CSS правил

При наданні оформлення елементу за допомогою відступів і рамок можна оперувати властивістю «кожної сторони» (зверху, праворуч тощо), а можна використовувати загальний термін (відступ, рамка), що приводить до значного скорочення css- правила. Але при скороченому записі правила **слід пам'ятати,** що «обхід»

проводиться за часовою стрілкою, тобто **зверху-праворуч-знизу-ліворуч**
Наприклад,

Розгорнутий запис	Скорочений запис

p {
border-style: dashed;
border-color: red;
border-width: 5px;
padding-top: 10px;
padding-right: 20px;
padding-bottom: 30px;
padding-left: 40px;
}
	p {
	/*для BORDER */
	/*додержуйтеся порядку : товщина тип колір */
	border: 5px dashed red;
	
	/*для PADDING / MARGIN */
	/* важливо додержуватися порядку при записі меж */
	padding: 10px 20px 30px 40px;
	}

Варіанти підключення CSS стилів

Внутрішні стилі:

in-line стилі. Додаємо оформлення безпосередньо у відкритому тегі елемента, за допомогою атрибуту style:

```
<header style="background-color: grey;">
    <p style="font-size: 20px;">
```

додавання стилю в елемент <head> за допомогою елементу <style> …. </style>:

```
<style type="text/css">
    header { background-color: grey; }
    p { font-size: 20px; }
</style>
```

Зовнішні стилі:

Підключення файлу стилів (style.css) за допомогою тега **<link>,** який включається у елемент <head>.

Створимо файл стилів за допомогою Sublime Text 3:

1. У меню File оберемо New File, потім Save with Encoding (де клікнимо UTF-8)
2. Збережемо файл з ім'ям style.css , а зі списку типів оберемо тип CSS.

У файл style.css запишемо і збережемо CSS-код, *наприклад*:

```
header { background-color: grey; }
p { font-size: 20px; }
```

У html-коді сторінці, до якої додаємо стилеве оформлення, в елементі <head> запишемо:

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

Імпортування стиля з файлу .css за допомогою елементу @import:

```
<style type="text/css">
        @import url("style.css")
</style>
```

Може показатися, що імпортування стилю нічим не відрізняється від підключення стилю, але це не так. Відмінності можуть проявитися, якщо до сторінки слід додати декілька стилевих оформлень (декілька файлів .css), спосіб за допомогою елементу @import може визивати помилки.

Зважаючи на викладене, більш прийнятним для використання є підключення зовнішнього файлу стилів за допомогою тега <link>. І такі зовнішні стилі ще називають «*зв'язані стилі*».
До того ж роздільне збереження файлу HTML розмітки і файлу CSS оформлення web- сторінки робить проект більш читабельним і гнучким. Файл оформлення можна використовувати для всіх сторінок власного сайту, зменшуючи час на розробку сайту.

Порядок застосування стилів при завантаженні web-сторінки у браузер:

1. дефолтні (по замовчуванню) стилі браузера;
2. стилі зовнішніх файлів, підключених до сторінки за допомогою тега <link>, та стилі, прописані всередині тега <style>, що міститься в <head>;
3. in-line стилі, що прописані всередині HTML елемента в <body>.

Успадкування і каскадування стилів.

Успадкування – це перенос правил оформлення, що задані для батьківського елемента, на дочірні елементи (такі, які містяться всередині батьківського). *Наприклад*, якщо задати колір елементу <body>, то цей колір набудуть і елементи h1, h2, p тощо, які містяться у <body>.
Не всі властивості можуть унаслідуватися, тому що це може не мати сенсу.
Каскадування – це механізм, якій керує остаточним результатом, якщо до одного елементу застосовуються кілька правил, які конфліктують між собою.
Надамо пріоритет селекторів при каскаду ванні

Місце розташування стилю або селектор	***Умовний пріоритет***
'!important'	перекриває всі попередні стилі
in-line стиль	+1000
id селектор	+100
class/pseudo class/attribute селектори	+10
tag селектор	+1
Browser default	0

Селектори

Селектор – це певне ім'я стиля, зв'язане із елементом з html-коду, до якого застосовується стиль. У якості селекторів виступають назви тегів, ідентифікаторів, класів, атрибутів тощо. Вміння правильно застосовувати селектори – ключове вміння верстальника сайтів.

Селектори елементів

Один і той же елемент (тег) у html- коді може зустрічатися декілька разів, *наприклад*, текст складається із абзаців, кожен абзац оформлений тегом <p>. Якщо всі абзаци мають однакове стилеве оформлення, то у якості селектора виступає ім'я тега (у нашому прикладі p):

```
p { font-size: 20px;}
```

Селектор-тег – це ім'я тега

Але, *наприклад*, якомусь одному з абзаців треба надати зелений колір. Як це зробити? Для цього абзац у html- коді слід «відмітити» (ідентифікувати) для того, щоб можна було використати цей ідентифікатор при наданні стилю. А саме, у відкритому тегі абзаца надати атрибут id

```
<p id="greenColor">Текст абзацу</p>
```

Тоді у css можна записати:

```
#greenColor {color: green;}
```

Селектор-идентифікатор – це им'я ідентифікатора (id), перед яким стоїть **хештег** «#»

Ідентифікатором відмічають унікальні (!) елементи сторінки.

Але не поспішайте використовувати цей селектор – він надто негнучкий, його важко застосовувати повторно у різних проектах. Намагайтеся замінити його селектором тегу, або навіть псевдокласу.

Для групування різних елементів, але таких, що мають щось спільне, *наприклад*, заголовки різного рівня і певний блок, використовують атрибут class:

```
<h1 class="caption">Текст заголовку</h1>
<h2 class="caption">Текст заголовку</h2>
<div class="caption">Текст елементу</div>
```

Тоді можна використати ім'я класу у якості селектора для надання стилю css:

```
.caption { background-color: pink;}
```

Селектор-клас – це ім'я класу(class), перед яким ставиться крапка «**.**»

Класи можна комбінувати, додаючи до одного тегу кілька класів.

Треба пам'ятати, що імена ідентифікаторів і класів є регістрозалежними і повинні писатися виключно латиницею.

Ідентифікатор має більш високий пріоритет, ніж клас. Тому, якщо для будь-якого елемента буде вказано і клас, і ідентифікатор (що не суперечить принципам CSS), то застосований буде стиль ідентифікатора.
Щоб гнучко управляти стилем подібних елементів, в CSS введені **селектори атрибутів**. Вони дозволяють встановити стиль за присутністю певного атрибута тега або його значення. Багато тегів розрізняються за своєю дією в залежності від того, які в них використовуються атрибути. *Наприклад*, тег <input> може створювати кнопку, текстове поле і інші елементи форми всього лише за рахунок зміни значення атрибуту type. При цьому додавання правил стилю до селектора input застосує стиль одночасно до всіх створених за допомогою цього тега елементам.
Синтаксис селектора атрибуту:

```
Селектор[атрибут="значення"] { Оголошення правил стиля }
```

Наприклад,

```
input[type="text"] {background-color: yellow; }
```

Універсальним селектором є селектор зірочка «*****»

```
* { color: blue;}
```

Якщо блоки оголошення стилів співпадають для різних селекторів, ці селектори можна групувати. **Група селекторів** - це селектори, між якими стоїть кома «**,**»:

```
S1, S2, S3 {БЛОК ОГОЛОШЕННЯ СТИЛІВ}
```

Група селекторів

Щодо необхідності групування селекторів в середовищі професійних верстальників йдуть дебати. Деякі верстальники застерігають від використання груп селекторів, бо це може приводити до погіршення читабельності коду, порушувати логіку надання стилів. До групування слід звертатися, коли у коді є значні фрагменти однакових оголошень стилів.

Псевдоелементи і псевдокласи

У попередньому пункті ми розглянули, як можна застосовувати правила CSS до елементів HTML. Але на web- сторінці є елементи, яких не існує в html-коді розмітки, *наприклад*, перший рядок абзацу, останнє слово в абзаці, перша буква слова тощо. Такі елементи називають ***псевдоелементами***. Псевдоелементам також можна задавати стиль, як і елементам HTML.
::first-letter – визначає стиль першого символу в тексті елемента, до якого додається.
Наприклад, для першої літери кожного абзацу (тег параграф):

```
p:first-letter {
    font-family: "Times New Roman", Times, serif;   /* Гарнітура шрифту */
    font-size: 200%;       /* Розмір шрифту */
```

```
        color: red;   /* Червоний колір*/
    }
```

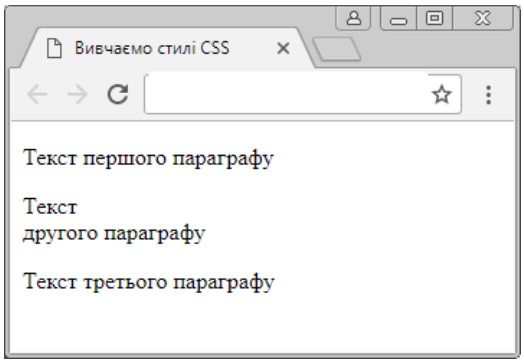
До цього псевдоелемента можуть застосовуватися тільки стильові властивості, пов'язані з властивостями шрифту, полями, відступами, рамками, кольором і фоном.

::first-line – задає стиль першому рядку форматованого тексту. Довжина цього рядка залежить від багатьох факторів, таких як шрифт, що використовується, розмір вікна браузера, ширина блоку, мови текстового контенту і т.ін.. У правилах стилю допустимо використовувати тільки властивості, що відносяться до шрифту, зміни кольору тексту і фону.

Наприклад,

```
p:first-line {
    color: red;                 /* Червоний колір тексту */
    font-style: italic;         /* Курсивне накреслення */
    font-weight: bold;          /* Жирне накреслення */
}
```

Розглянемо *приклад*.



Створимо web-сторінку

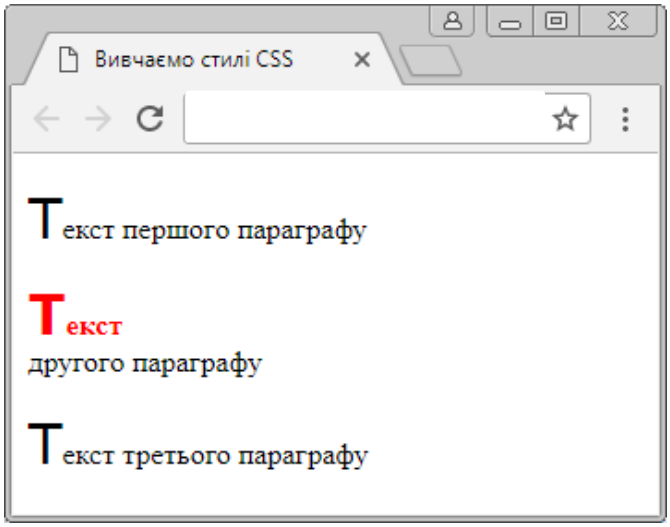
Код сторінки складається із трьох параграфів, у другому параграфі додамо ідентифікатор id=”Sel”, а після слова «Текст» вставимо перехід на новий рядок <br>, а саме:

```
<p>Текст першого параграфу</p>
<p id=”Sel”>Текст<br>другого параграфу</p>
<p>Текст третього параграфу</p>
```

Додамо до цієї сторінки стилі:

```
p::first-letter {
    font-family: Verdana;
    font-size: 30px;
}
#Sel::first-line {
    color: red;
    font-weight: bold;
}
```

І у браузері побачимо



Псевдокласи – це ключові слова, які додаються до селектора і визначають його особливий стан. Псевдокласи визначають динамічний стан елемента, якій змінюється під дією користувача, і положення у ієрархічному дереві елементів.

Наведемо **псевдокласи**, для визначення сусідніх елементів у ієрархії батьки-нащадки:

:first-child – перший з нащадків свого батька.;
:last-child – останній з нащадків свого батька;
:only-child - єдиний нащадок свого батька, сусідніх елементів немає.
:nth-child(k) – нащадок номер k свого батька (нумерація починається з 1), наприклад, :nth-child(2) – другий нащадок;
:nth-child(kn+b) – розширення попереднього селектору, де номер нащадка визначається формулою «kn+b», тут k і b – константі, а у якості n мається на увазі будь-яке число. *Наприклад*, :nth-child(2n) – всі нащадки з парними номерами (2, 4, 6, …), а :nth-child(2n+1) – з непарними (1, 3, 5, …)

Можна визначити псевдокласи, які враховують не всіх сусідів, а **тільки сусідів із вказаним тегом**:
:first-of-type
:last-of-type
:only-of-type
:nth-of-type
:nth-last-of-type
Вони аналогічні до псевдо класів, наведених вище, але відфільтровують тільки нащадків із вказаним типом елементів.

За допомогою псевдокласів можна отримати динамічні ефекти на сторінці. Використовуючи той факт, що браузер певний час зберігає історію дій користувача на сторінці, розглянемо приклад використання псевдокласів до елементу a (посилання):

:link – посилання, яке не відвідувалося користувачем;

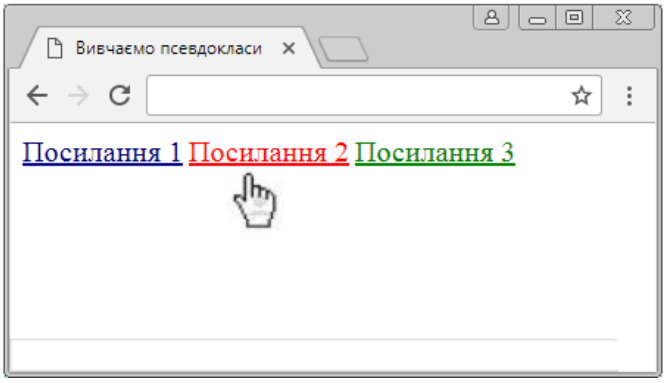
:visited - посилання, яке було відвідане користувачем, тобто користувач клікав мишею по посиланню

:hover – звертання до посилання, на яке користувач наводить мишею(без кліка);

:active – активне посилання, таке що виконується у даний момент.

Створить наступний приклад і розгляньте його дію у браузері:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Вивчаємо  псевдокласи</title>
                <meta charset="utf-8">
                <style type="text/css">
/* колір посилань, які не відвідувалися */
        a:link { color: navy; }
/* колір посилань, які були відвідані */
        a:visited { color: green; }
/* колір посилань, на які наводиться курсор миші */
        a:hover { color: red; }
/* колір активних посилань */
        a:active { color: yellow; }
                </style>
        </head>
        <body>
                <a href="1.html">Посилання 1</a>
                <a href="# ">Посилання 2</a>
                <a href="# ">Посилання 3</a>
        </body>
</html>
```



Додаткові знаки у селекторах і прийоми використання селекторів

Наведемо рекомендацію Джеффрі Уея (Jeffrey Way, керівник курсів web-розробки Tuts +, за матеріалами сайту www.net.tutsplus.com) щодо селекторів і прийомів їх застосування, «які повинен знати кожен верстальник»:

Універсальний селектор «*» (зірочка) – виділяє кожен елемент на сторінці. Найчастіше його використовують для «зкидання» всіх полів і відступів у всіх елементах, *наприклад*:

```
* {
margin 0px;
padding 0px;
}
```

Також зірочку можна використовувати для означення дочірніх елементів об'єкту, наприклад для об'єкту із ідентифікатором id="container":

```
#container * {
 border: 1px solid black;
}
```

Селектор нащадка S T – його слід використовувати, коли проводиться більш «точковий» вибір елементу, *наприклад*, обрати тільки ті тегі посиланнь, що є у списках (всіх a-нащадків елемента li):

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

Але не захоплюйтесь «вкладеністю» нащадка – запис *X Y Z A B.title явно потребує спрощення* ☺

Псевдокласи (використовуємо «**:**» двокрапку), *наприклад*,

обрати посилання, на які ще не проводився клік мишею — a:link
обрати посилання, на які наводиться курсор миші — a:hover
обрати посилання, за якими було здійснено клік або перехід — a:visited

```
a:link {color: blue;}
div:hover {color: yellow;}
a:visited {color: purple;}
```

відмічена радіо-кнопка (аналогічно для «флажка»):

```
input[type="radio"]:checked {
   border: 1px solid black;
}
```

Суміжний селектор S + R (використовуємо знак «+»), *наприклад*, кожен наступний параграф, який йде після маркованого списку буде зелений:

```
ul + p {
  color: green;
}
```

S > T – обрати прямого нащадка

Різниця між «S T» і «S > T» в тому, що останній вибере тільки прямих нащадків.

Комбінація сіблінгових (сестринських) **елементів S ~ T** («~» знак тильда). Схожа на **S + T,** але менш сувора, *наприклад* порівняйте,

ul ~ p {
color: red;
}
	Всі параграфи після маркованого списку будуть червоні

ul + p {
color: red;
}
	Тільки перший параграф, що йде за маркованим списком буде червоний

S[attribute] селектор атрибуту елемента, *наприклад*,

```
input[type="text"] {background-color: yellow; }
```

Робота із атрибутом **href у посиланнях:**

- надання стилю тільки посиланню із конкретно вказаною адресою (http://net.tutsplus.com):

```
a[href="http://net.tutsplus.com"] {
  color: green;
}
```

- додаємо знак «*», який означає, що шукане значення (tuts) може перебувати в будь-якій частині атрибута href:

```
a[href*="tuts"] {
  color: green
}
```

- додаємо знак «^» (карат) - це «нагадування» користувачеві про те, що посилання веде на інший сайт. Поряд із посиланням буде розміщено невеличку іконку-нагадування:

```
a[href^="http"] {
  background: url(path/to/external/icon.png) no-repeat;
  padding-left: 10px;
}
```

- додаємо знак «$» (долар) – посилання на кінець строки, *наприклад*, шукаємо всі посилання, що містять картинки тип .jpg:

```
a[href$=".jpg"] {
  color: red; }
```

Виключення (заперечення) селектора X:not(selector)

Припустимо, що треба обрати всі тегі, крім тега <p>^

```
*:not(p) {
  color: green;
}
```

або всі блоки , крім того, що має ідентифікатор id=”container”:

```
div:not(#container) {
  color: blue;
}
```

Розглянемо *приклад*, припустимо, що в нас є код:

```
<div>
  <p>Тут параграф.</p>
  <ul>
    <li>Елемент 1.</li>
    <li>Елемент 2.</li>
  </ul>

  <ul>
    <li>Елемент 3.</li>
    <li>Елемент 4.</li>
  </ul>
</div>
```

І ми хочемо встановити стиль для «Елемент 2». Зробити це можна кількома засобами, а саме:

Варіант 1. Базується на обиранні прямого нащадка.

Знайди перший маркований список на сторінці, потім знайди тільки його прямих нащадків, які є елементами li. Після цього вибери тільки другий по порядку елемент li:

```
ul:first-of-type > li:nth-child(2) {
  font-weight: bold;
}
```

Варіант 2. Базується на суміжних селекторах.

Знаходимо ul, наступний безпосередньо за тегом параграфа, після чого знаходимо найостанніший його дочірній елемент:

```
p + ul li:last-child {
  font-weight: bold;
}
```

Варіант 3.
Отримуємо перший елемент ul на сторінці, потім шукаємо найперший елемент li, але починаючи з кінця:

```
ul:first-of-type li:nth-last-child(1) {
  font-weight: bold;
}
```

Властивості CSS правил

Оформлення текстів

Накреслення шрифтів

Жирне накреслення	**font-weight : bold;**
Курсивне накреслення	**font-style: italic;**
Підкреслений текст	**text-decoration: underline;**
Надкреслений текст	**text-decoration: overline;**
~~Закреслений~~ текст	**text-decoration: line-through;**

Всі властивості мають значення none, яке переводить властивість у стан normal

Наприклад,

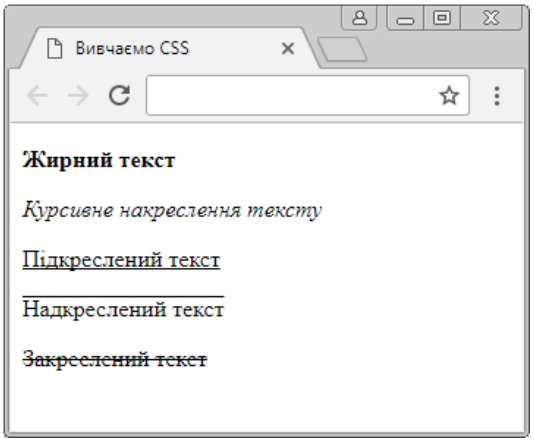
Для сторінки

```
<body>
        <p id="_bold">Жирний текст</p>
        <p id="_italic">Курсивне накреслення тексту</p>
        <p id="_underLine">Підкреслений текст</p>
        <p id="_overLine">Надкреслений текст</p>
        <p id="_lineThrough">Закреслений текст</p>
</body>
```

Можна додати css правила:

```
#_bold {font-weight : bold;}
#_italic {font-style: italic;}
#_underLine {text-decoration: underline;}
#_overLine {text-decoration: overline;}
#_lineThrough {text-decoration: line-through;}
```

І побачити



Розмір шрифту надається властивістю **font-size** , значення якої прийнято вказувати у пунктах (але можна і інші одиниці виміру), *наприклад*

```
p {font-size: 18pt;}
```

Відступ першого рядка абзацу **text-indent**

Наприклад відступ 1,5см

```
p { text-indent: 1.5cm;}
```

Міжрядковий інтервал **line-height**

Наприклад полуторний інтервал,

```
p { line-height: 1.5em;}
```

Вирівнювання тексту в боксі **text-align**

Має такі значення:

посередині	**center**
ліворуч	**left**
праворуч	**right**
по ширині	**justify**

Сімейства шрифтів **font-family**

За допомогою цієї властивості можна задати тип шрифта, наприклад Arial,

```
p { font-family: Arial;}
```

Але не має ніякої гарантії, що зазначений у правилі шрифт, буде існувати на тому пристрої, де буде відображатися сторінка. Тому при роботі із типами шрифтів вказують кілька шрифтів з сімейства. Наведемо імена базових сімейств і приклади шрифтів, які відносяться до кожного:

- **serif** – антиква, шрифти із «засічками» : Times New Roman, Bookman Old Style, Garamond, Georgia
- **sans-serif** – гельветика, «без засічок» або «рублені» : Arial, Tahoma, Verdana, Century Gothic
- **monospace** – моноширінні, з однаковою шириною знакомісця для кожного символу: Courier, Courier New, Lucida Console

Для того, щоб шрифт був гарантовано найдений слід у якості значення властивості font-family записати кілька типів шрифтів (розділитель кома «,»), а наприкінці вказати ім'я сімейства.

Наприклад,

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

Розріджений чи ущільнений текст , тобто відстань між символами, можна задати за допомогою властивості **letter-spacing**

Наприклад,

```
span { letter-spacang: 3px;}
```

Колір шрифту задається властивістю **color**, наприклад,

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

Використання власних шрифтів. Правило @font-face

Створюючи дизайн web- сторінок можна використовувати і нові шрифти, колекції яких у Інтернеті повсякденно поповнюються.

Правило **@font-face {ВЛАСТИВОСТІ ШРИФТУ}** дозволяє визначити настройки шрифтів, а також завантажити специфічний шрифт на комп'ютер користувача. Всередині конструкції @font-face може перебувати набір властивостей для зміни параметрів шрифту (такі як font-family, font-size, font-style і ін.), а також посилання на файл специфічного шрифту. Посилання записується у вигляді

src: url (URL) ,

де URL - відносний або абсолютний шлях до файлу шрифту.

Наприклад, в нас є специфічний шрифт pompadur.ttf , якій знаходиться у папці Font нашого комп'ютера. І ми хочемо надати цей шрифт до елементів параграф web-сторінки. Тоді у css-правилах слід написати:

```
@font-face {
        font-family: Pompadur;              /* дамо ім'я нашому шрифту */
        src: url(Font/pompadur.ttf);        /* вкажімо адрес файла зі шрифтом */
}
p {
          font-family: Pompadur;            /* надамо наш шрифт до селектору */
}
```

Управління фоном

Кольоровий фон блочних елементів:

- **background: color** – задається колір фону

```
body {background: pink;}
```

Можна задавати **градієнтний фон**. Градієнтний колір – це переходи від одного кольору до іншого, задаються за допомогою функцій **linear-gradient()** та **radial-gradient().**

Лінійний градієнт, *наприклад,*

```
background: linear-gradient(45deg, yellow, green);
```

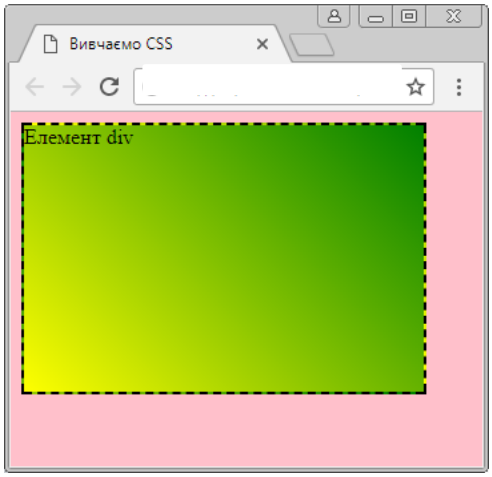
де 45deg - це кут нахилу, що вимірюється у градусах, по якому перший колір (yellow) переходить у другий (green). Замість градусної міри можна використовувати ключові слова:

- to top – відповідає 0deg
- to right– відповідає 90deg
- to bottom– відповідає 180deg
- to left– відповідає 270deg

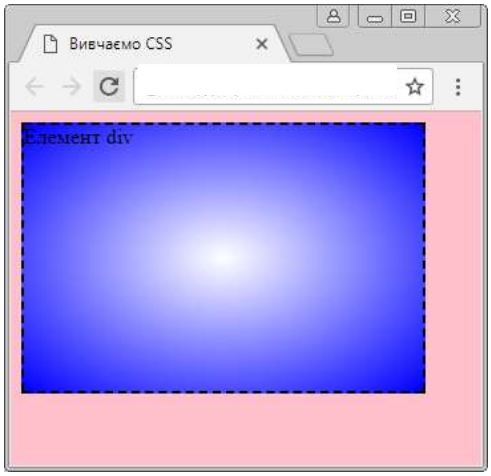
Радіальний градієнт можна записати, *наприклад*,

```
background: radial-gradient(white, blue);
```

у наведеному прикладі місце розташування центру радіального градієнту (перший атрибут функції radial-gradient(), за аналогією із функцією linear-gradient()) не вказаний, і за замовчуванням він є at center.



лінійний градієнт



радіальний градієнт

Фоном елементу можна зробити завчасно збережений малюнок.

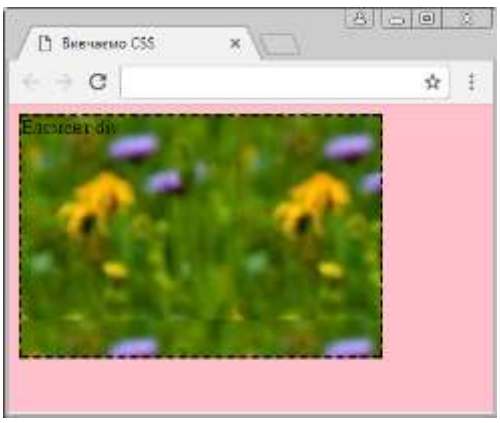
- **background-image: url(“ім’я малюнку”)** – для фона задається малюнок.

Наприклад, в нас є малюнок розміром 180×170 пікселів, збережений у файлі Галявина.jpg

запишемо у стилі

```
background-image: url("Галявина.jpg");
```

і отримаємо



Для елементу div був заданий розмір width: 300px; height: 200px; тому наш малюнок був повторений. Якщо не задані інші властивості браузер повторює картинку, доти не буде «замощений» увесь бокс елементу.

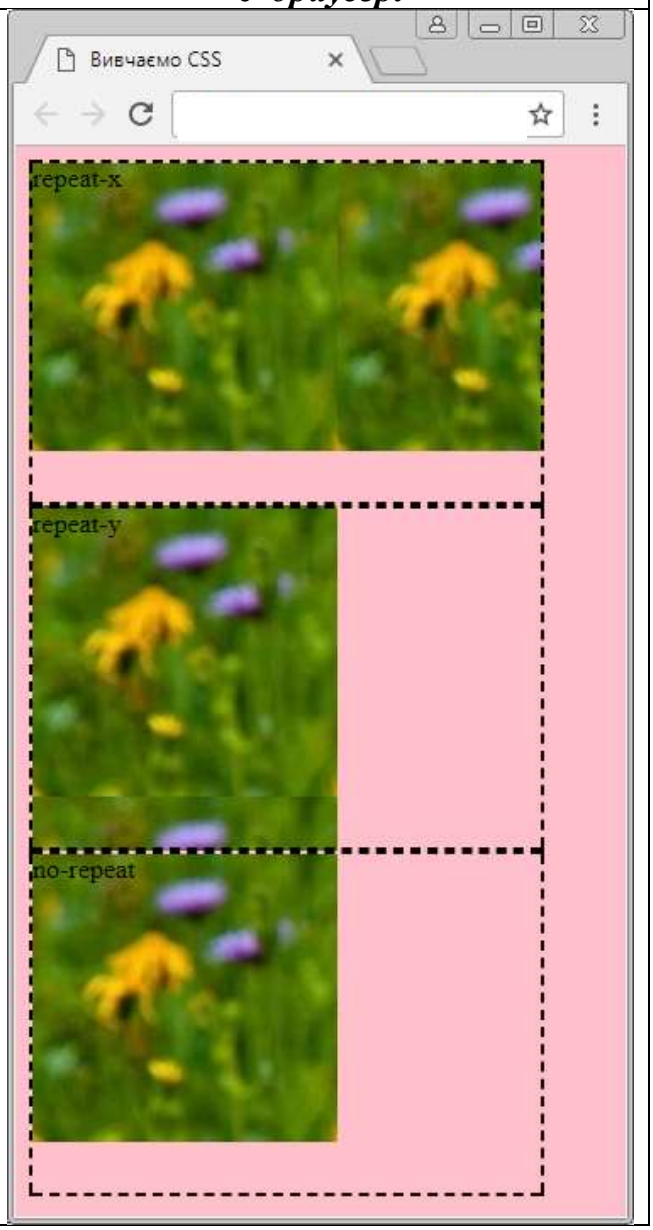
Управління «замощуванням» фонового зображення

Тип «замощування» картинкою бокса елементу можна міняти за допомогою властивості **background-repeat**, яка має такі значення:

- **repeat** – повторення малюнка по вертикалі й горизонталі;
- **repeat-x** - повторення малюнка виключно по горизонталі;
- **repeat-y** - повторення малюнка виключно по вертикалі;
- **no-repeat** – заборона повторення.

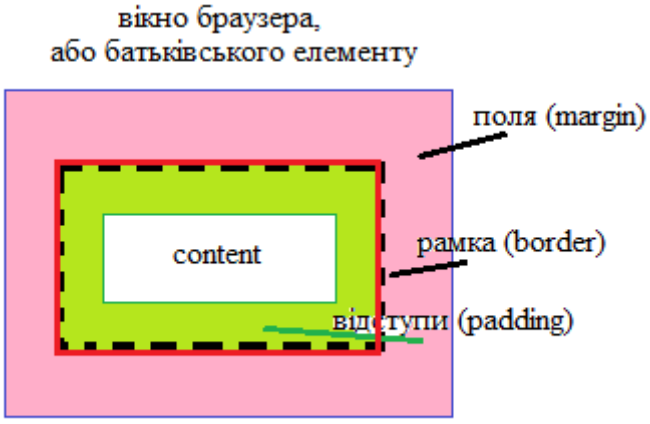
По замовчуванню властивість background-repeat встановлена із значенням repeat.

html код зі стилем	*У браузері*

<html>
<head>
<style type="text/css">
body {background:pink;}
div {
width: 300px;
height: 200px;
border: 2px dashed black;
background-image:
url("Галявина.jpg");
}
#_rX { background-repeat: repeat-x; }
#_rY { background-repeat: repeat-y; }
#_rN { background-repeat: no-repeat; }
</style>
</head>
<body>
 repeat-x
 repeat-y
 no-repeat
</body>
</html>
	

Управління розташуванням блочних елементів

Всі блочні елементи надані у CSS комбінацією правил полів (margin) , рамки (border), відступів (padding) і вмістом (контентом / content)



Рамка характеризується такими параметрами:

- товщиною (властивість border-width);
- типом (властивість border-style);
- кольором (властивість border-color);

Властивість border-style може приймати одне із значень:

Властивість **border-width** характеризують три значення, які задають ширину кордону рамки:

- **thin** - 2 пікселя;
- **medium** - 4 пікселі;
- **thick** - 6 пікселів

Наприклад,

```
border-width : medium;
```

Для більш точного значення, товщину можна вказувати в пікселях або інших одиницях. Дефолтне значення thin.

Кожну властивість рамки можна записувати окремим правилом, *наприклад*,

```
p {
    border-style: dashed;
    border-color: red;
    border-width: 5px;
}
```

А можна використати скорочений запис, але при скороченому записі треба додержуватися порядку перерахування параметрів:

BORDER : товщина тип колір;

Наприклад,

```
p {border:  5px dashed red;}
```

Властивість box-sizing

Застосовується для зміни алгоритму розрахунку ширини і висоти елемента. Згідно зі специфікацією CSS ширина блоку складається з ширини контенту (width), значень полів (margin), відступів (padding) і рамки (border). Аналогічно є і з висотою блоку. Властивість **box-sizing** дозволяє змінити цей алгоритм, щоб властивості width і height задавали розміри не контента, а розміри блоку.
Синтаксис

box-sizing: content-box | border-box | padding-box | inherit

тут

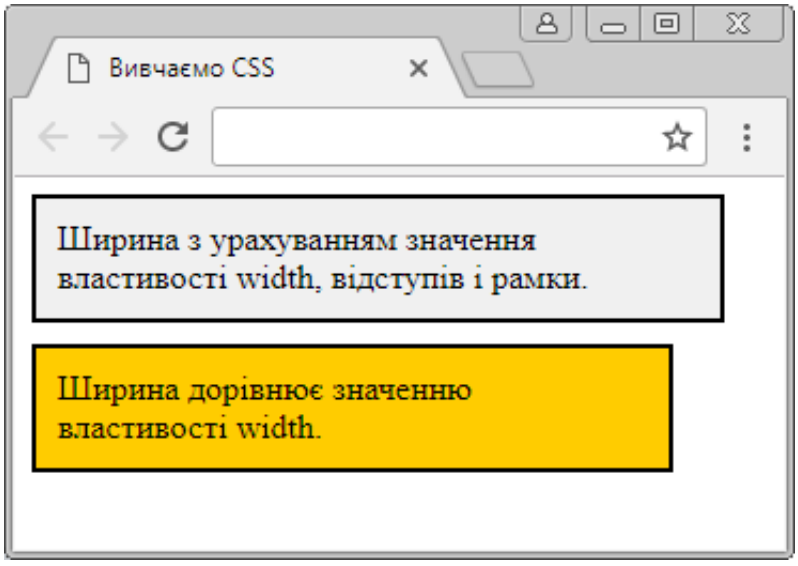
- **content-box** – ґрунтується на стандартах CSS, при яких властивості width і height задають ширину і висоту контенту і не містять в собі значень полів, відступів і рамки;
- **border-box** - властивості width і height містять у собі значення рамки (border-width) і відступів (padding) , але **не** полів (**margin**)
- **padding-box** - властивості width і height містять у собі значення відступів (padding) , але **не** рамки (**border-width**) і полів (**margin**)
- **inherit** – успадковує значення батьківського елементу

Дефолтне значення властивості box-sizing є content-box.

Розглянемо *приклад*,

```
<!DOCTYPE html>
<html>
        <head>
                <title>Вивчаємо  CSS</title>
                <meta charset="utf-8">
                <style type="text/css">
.box1 {
   background: #f0f0f0;          /* колір фона */
   width: 300px;                 /* ширина блока */
   padding: 10px;                /* відступи */
   border: 2px solid #000;       /* параметри рамки */
}
.box2 {
   background: #fc0;             /* колір фона */
   width: 300px;                 /* ширина блока */
   padding: 10px;                /* відступи */
   margin-top: 10px;             /* поле зверху */
   border: 2px solid #000;       /* параметри рамки */
   box-sizing: border-box;       /* ширина блоку з відступами */
}
                </style>
        </head>
        <body>
                 <div class="box1">
Ширина з урахуванням значення властивості width, відступів і рамки
                </div>
                <div class="box2">
Ширина дорівнює значенню властивості width.
                </div>
        </body>
</html>
```

У браузері побачимо



В даному прикладі ширина першого шару буде дорівнює 324 пікселя, оскільки вона складається зі значення ширини контенту (width = 300), відступів зліва і справа (padding = 10+10) і товщини кордонів рамки (border = 2+2).
Ширина другого шару дорівнює 300 пікселів за рахунок застосування властивості box-sizing.

Властивості ширина (товщина) рамки (border-width), поля (margin) і відступи (padding) можна задавати одним, двома, трьома або чотирма значеннями
Наприклад, товщина рамки може бути задана одним із наведених прикладів:

```
border-width: 3px;
border-width: 3px 7px;
border-width: 3px 7px 7px;
border-width: 3px 7px 7px 4px;
```

Від кількості значень залежить результат встановлення товщини на межах рамки, дивись таблицю

Число значень	**Результат**
1	Товщина межі рамки буде встановлена для усіх сторін елемента.
2	Перше значення встановлює товщину верхньої (top) і нижньої (button) межі, друге — лівої (left) і правої (right).
3	Перше значення встановлює товщину верхньої (top) межі , друге — одночасно лівої (left) та правої (right) межі, а третє — нижньої (button) межі.
4	Почергове встановлює товщину верхньої (top), правої (right), нижньої (button) і лівої (left) меж рамки.

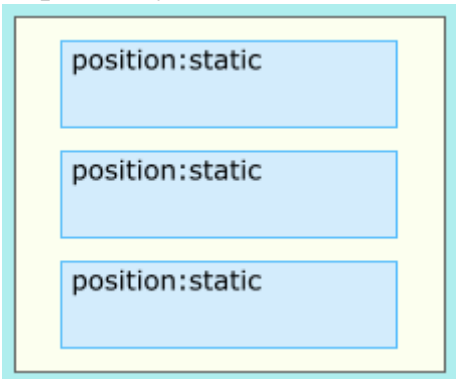
*За аналогією діють і параметри полів (*margin*) і відступів (*padding*).*

Позиціювання, z-index і «плаваючі» елементи. Flexbox

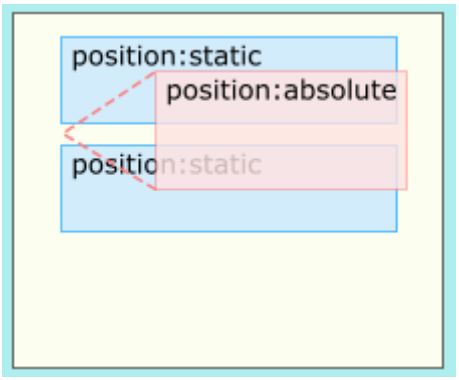
CSS-позиціювання (*positioning)* дозволяє вказати, де з'явиться блок елемента, а вільне переміщення (*floating*) переміщує елементи до лівого або правого краю блоку-контейнера, дозволяючи решті вмісту «обтікати» його.

Типи позиціювання, властивість position
Властивість position дозволяє точно задати нове місце розташування блоку щодо того місця, де він знаходився б в нормальному потоці документа. За замовчуванням всі елементи розташовуються послідовно один за іншим в тому порядку, в якому вони визначені в структурі HTML-документа. Властивість не успадковується.
Дефолтним значення властивості є **static**. Таке розташування елементів носить назву «прямий потік».

position: **static** – природне розташування елементів, прямий потік:



position: absolute – абсолютне позиціювання видаляє елемент з прямого потоку позиціювання і дозволяє розмістити його на сторінці довільним чином, вказавши координати:



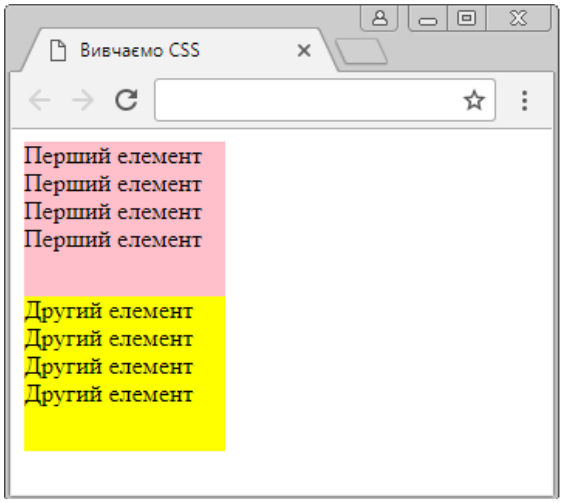
Для завдання координат використовують властивості **top** (зверху) і **left** (ліворуч). **Координати задаються відносно лівого верхнього кута** блоку, що вміщує, тобто блоку, в якому міститься позиціонуємий елемент. У найпростішому випадку позиціонується щодо лівого верхнього кута сторінки.

Для відліку координат **від правого нижнього кута** використовуються властивості **bottom** (знизу) і **right** (праворуч).
Розглянемо *приклад*
Без позиціювання
html- код:

```
<head>
    <style type="text/css">
#box1 {
    width: 130px;
    height: 100px;
    background-color: pink;
}
#box2 {
    width: 130px;
    height: 100px;
```

```
        background-color: yellow;
}
        </style>
</head>
<body>
         <div id="box1">Перший елемент Перший елемент Перший елемент
Перший елемент</div>
        <div id="box2">Другий елемент Другий елемент Другий елемент
Другий елемент</div>
</body>
```

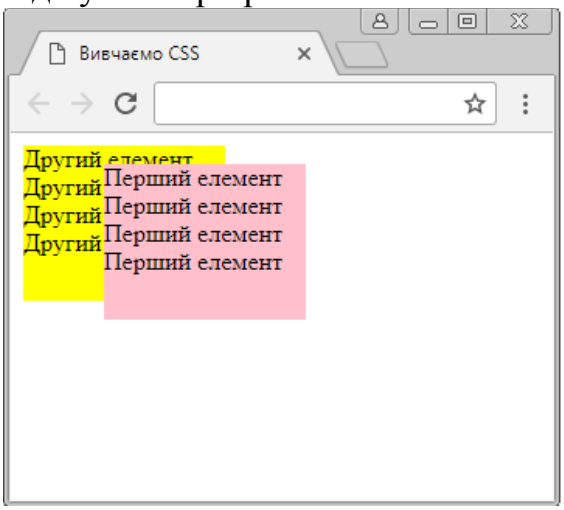
У браузері



Додамо для першого елементу **абсолютне позиціювання**:

```
        position: absolute;
        top: 10px;
        left: 60px;
```

Ці властивості слід вказувати першими у блоці оголошень властивостей селектору #box1, беручі до уваги пріоритети.

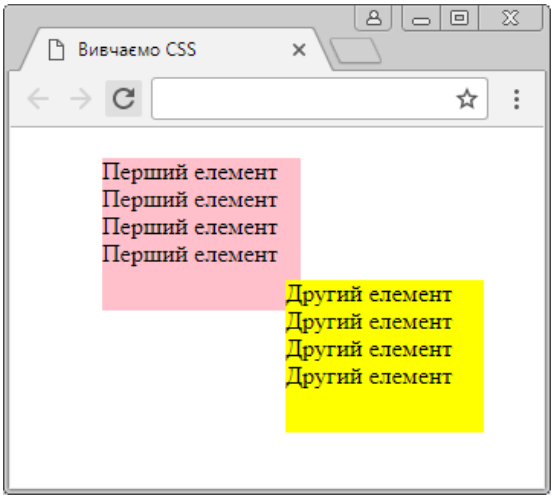


Як бачимо, «Перший елемент» вийшов із потіку, розташувався так, що його лівий верхній кут став у координату зверху 20, а зліва 60 пікселів, відраховану від вікна браузеру, а його місце у потоці зайняв «Другий елемент».

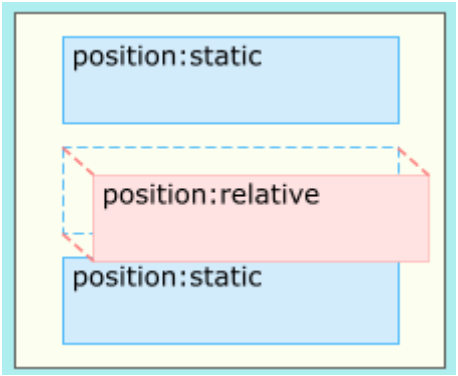
Задамо абсолютне позиціювання і «Другому елементові», тобто додамо у селектор #box2 таке:

```
position: absolute;
top: 100px;
left: 180px;
```

На сторінці побачимо

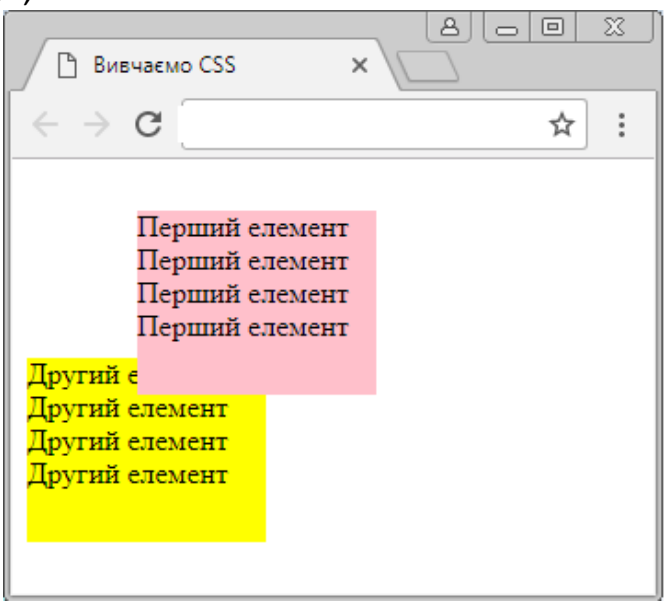


position: relative – відносне позиціювання - елемент можна зрушити щодо того місця, де він був би в потоці, але при цьому з потоку він не виключається, а замість нього буде пусте місце. Тобто зсувається зі свого місця він тільки візуально, а положення всіх елементів навколо нього ніяк не змінюється.



При відносному позиціонуванні точкою початку відліку координат елемента є його стандартна позиція, обчислена браузером автоматично. Координати елемента top і left визначають зміщення елемента, щодо його стандартного положення. При указанні тільки однієї властивості position: relative, елемент не переміститься, так як спочатку знаходиться на початку координат.
У попередньому прикладі приберемо позиціювання у «Другому елементі», а для «Першого елементу» задамо (селектор #box1):

```
position: relative;
top: 20px;
left: 60px;
```



«Перший елемент» змістився відносно свого попереднього розташування, а не від розташування, відносно вікна браузеру, а «Другий елемент» залишився на своєму місці.

position: fixed – фіксоване позиціювання фіксує елемент в зазначеному місці сторінки. Блоком-контейнером фіксованого елемента є вікно перегляду, тобто елемент завжди фіксується щодо вікна браузера і не змінює свого положення під час прокрутки сторінки. Сам елемент при цьому повністю видаляється з основного потоку документа і створюється в новому потоці документа.

Властивість z-index

Якщо Ви маєте в своєму розпорядженні кілька елементів поруч на web-сторінці, то вони можуть перекриватися (див. приклади позиціювання), і важливо передбачити порядок їх перекриття.

Зазвичай елементи розташовуються відповідно їх визначенням у вихідному коді.

Елемент, визначений першим, буде розташовуватися надалі від нас, а елемент, заданий останнім, - найближче до нас. Найбільше значення дорівнює 9999.

Властивість z-index дозволяє управляти порядком розташування елементів.

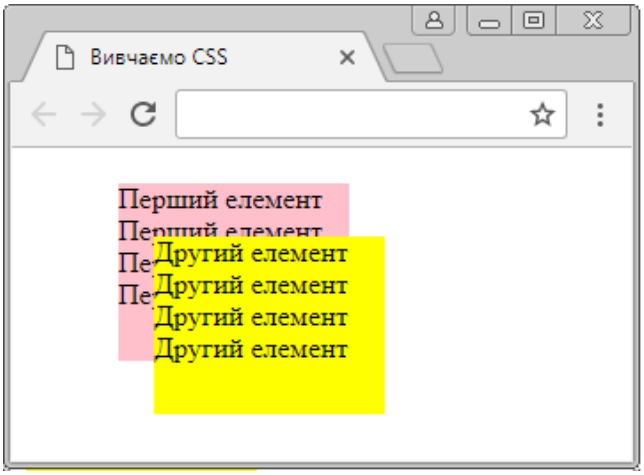
Привласнюючи різні цілочисельні значення властивості z-index, можна змінювати положення елемента в «стопці». Чим більше це значення, тим «вище» (ближче до нас) елемент.

Наприклад, розглянемо розташування елементів <body>

```
<div id="box1">Перший елемент Перший елемент Перший елемент Перший елемент</div>
<div id="box2">Другий елемент Другий елемент Другий елемент Другий елемент</div>
```

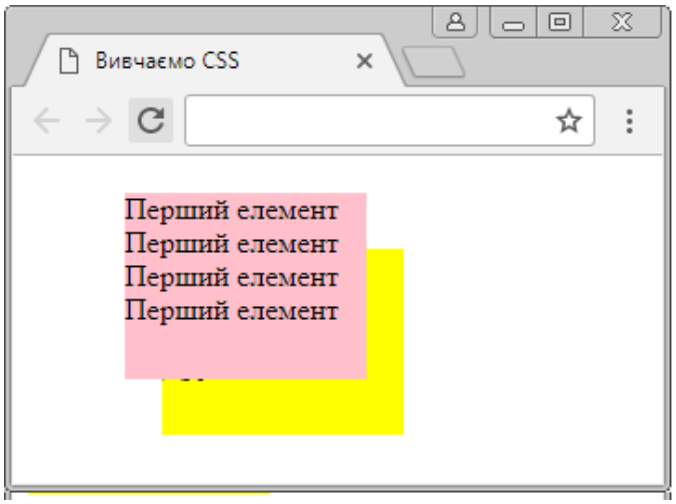
Стандартне перекриття

```
#box1 {
        position: absolute;
        top: 20px;
        left: 60px;
        width: 130px;
        height: 100px;
        background-color: pink;
}
#box2 {
        position: absolute;
        top: 50px;
        left: 80px;
        width: 130px;
        height: 100px;
        background-color: yellow;
}
```



Застосування z-index

```
#box1 {
        position: absolute;
        top: 20px;
        left: 60px;
        z-index: 20;
        width: 130px;
        height: 100px;
        background-color: pink;
}
#box2 {
        position: absolute;
        top: 50px;
        left: 80px;
        z-index: 10;
        width: 130px;
        height: 100px;
        background-color: yellow;
}
```



За допомогою **float** ми створюємо так звані **«плаваючі» елементи**. Плаваючими будемо називати такі елементи, які обтікаються по контуру іншими об'єктами web- сторінки, *наприклад*, текстом.

Плаваючі елементи досить активно застосовуються при верстці і в основному служать для втілення таких завдань.

- обтікання картинок текстом;
- створення врізок;
- розташування шарів по горизонталі (додавання колонок).

Все це виконує одна стильова властивість

float: left | right | none

- **left –** елемент розташовується зліва;
- **right –** елемент розташовується справа;
- **none –** обтіканіе не задано.

Властивість float має сенс тільки тоді, коли у елемента задана ширина width.

Вказавши для одного елементу властивість float, **ми всі наступні за ним елементи теж робимо плаваючими.**

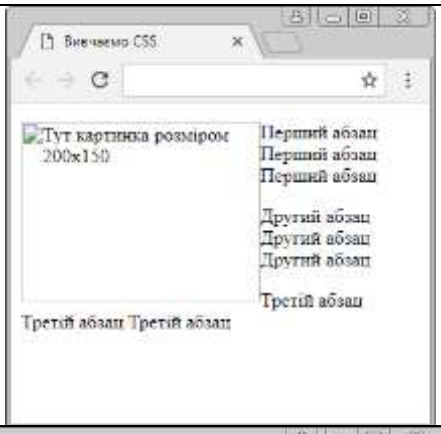
Наприклад,

Для наступного елемента <body>:

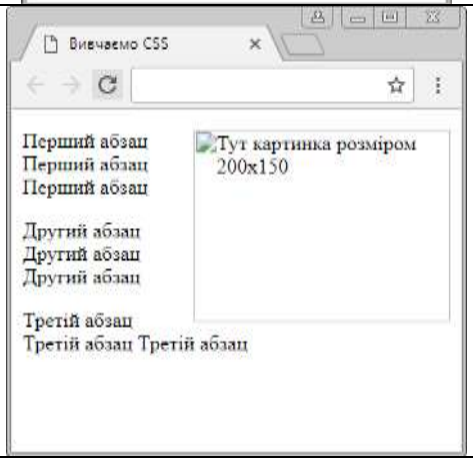
```
<body>
    <img src="#" alt="Тут картинка розміром 200x150">
    <p>Перший абзац Перший абзац Перший абзац</p>
    <p id="only">Другий абзац Другий абзац Другий абзац</p>
    <p>Третій абзац Третій абзац Третій абзац</p>
</body>
```

надамо стилі:

```
img {
    width: 200px;
    height: 150px;
    float: left;
}
```




```
img {
    width: 200px;
    height: 150px;
    float: right;
}
```



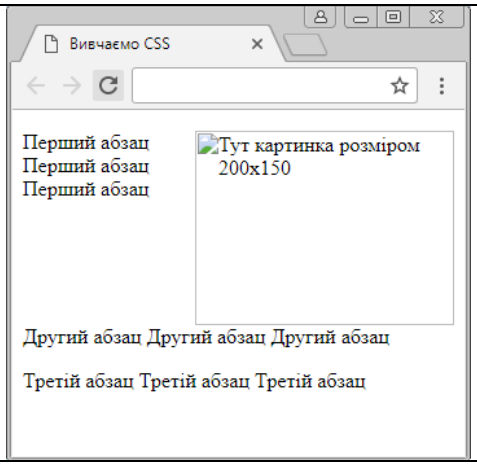
Всі елементи будуть плаваючими до тих пір, поки обтікання не буде скасовано властивістю **clear**.

Властивість **clear** має значення: **left, right, none або both**.

Ця властивість вказується в тому елементі, починаючи з якого ми хочемо скасувати плаваючі елементи. Тоді цей і наступні елементи почнуть розташовуватися нижче плаваючих елементів.

Наприклад

```
img {
        width: 200px;
        height: 150px;
        float: right;
}
#only {
        clear: right;
}
```



Приклад «плаваючої» верстки

Для елемента <body>:

```
<body>
        <header>Заголовок документу</header>
        <aside>Зміст документу</aside>
        <main>Основний вміст документу</main>
        <footer>Copiriating</footer>
</body>
```

Запишемо css правила:

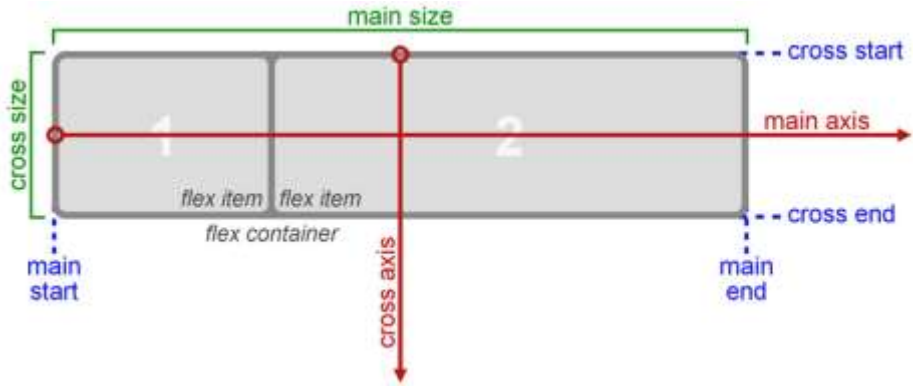
```
header, aside, main, footer {display: block;}
header { width: 900px;  height: 100px;
                background: peachpuff; }
aside { width: 300px; height: 400px;
                float: left;
                background: pink; }
main { margin: 0px 0px 0px 300px; width: 600px; height: 400px;
                background: azure; }
footer { width: 900px; height: 100px;
                background: gold;
                clear: left; }
```

І отримаємо сторінку у браузері

Останнім часом набуває популярності розміщення блоків на сторінці за допомогою засобу **flexbox**.

Flexbox – це технологія, за допомогою якої зручно вирівнювати елементи по горизонталі і вертикалі, змінювати порядок відображення елементів, розтягувати блокі на всю висоту батьківського блока, або «прибивати» блоки до нижньої межі «батька».

Flexbox потребує іншого уявлення про розташування елементів у потоці, оперуючи поняттям «вісей»:



Основні переваги flexbox:

- Всі блоки дуже легко робляться «гумовим», що вже випливає з назви «flex». Елементи можуть стискатися і розтягуватися за заданими правилами, займаючи потрібний простір.
- Вирівнювання по вертикалі і горизонталі, базової лінії тексту працює якісно.
- Розташування елементів в html не має вирішального значення. Його можна поміняти в CSS.

- Елементи можуть автоматично вибудовуватися в кілька рядків / стовпців, займаючи все надане місце.
- Безліч мов в світі використовують написання справа наліво rtl (right-to-left), на відміну від звичного нам ltr (left-to-right). Flexbox адаптований для цього. У ньому є поняття початку і кінця, а не права і ліва. Тобто в браузерах з «написанням» rtl всі елементи будуть автоматично розташовані в реверсному порядку.
- Синтаксис CSS правил дуже простий і освоюється досить швидко.

Ознайомитися із цією технологією можна на сайті https://www.w3.org/TR/css-flexbox-1/ у блозі сайту http://html5.by/blog/flexbox/ .

«Переповнення» розмірів елементу. Властивість overflow

Якщо певному елементу задані властивості width або height, то можуть траплятися випадки, коли контент перевищує задані розміри елементу. Для обробки таких ситуацій існує властивість **overflow**.

Властивість **overflow** має чотири значення:

- **visible** - (дефолтне значення). Контент одного елемента може виводитися упереміш з контентом його сусідів;
- **hidden** - весь контент, який знаходиться за рамками елемента, буде прихований від перегляду;
- **auto** - у разі необхідності створюються смуги прокрутки (скроли), щоб зробити невидимий контент доступним відвідувачеві;
- **scroll** - в елемент будуть вбудовані вертикальна і горизонтальна смуги прокрутки, навіть якщо вони не потрібні.

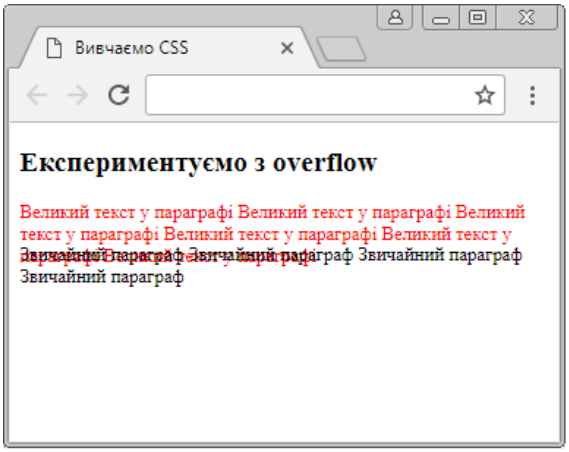
Розглянемо *приклад*. Створимо елемент <body>:

```
<body>
    <h2>Експериментуємо з overflow</h2>
    <p id="special">Великий текст у параграфі Великий текст у параграфі Великий текст у параграфі Великий текст у параграфі Великий текст у параграфі Великий текст у параграфі</p>
    <p>Звичайний параграф Звичайний параграф Звичайний параграф Звичайний параграф</p>
</body>
```

І додамо стиль ідентифікованому (id=special) параграфу, інші елементи мають дефолтні значення

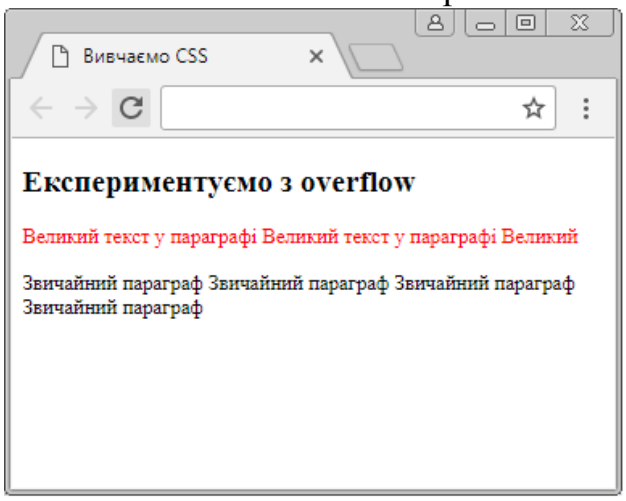
```
#special {
    height: 20px;
    color: red;
    overflow: visible;
}
```

У браузері побачимо:



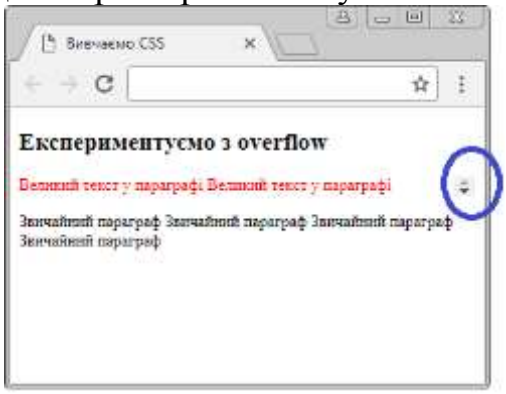
Ми могли і не вказувати overflow: visible; за замовчуванням текст елементу #special , якому ми обмежили висоту до 20px, «наліз» на наступний за ним елемент.

Змінимо значення visible на значення **hidden** і отримаємо:



Решта «Великого тексту», яка перевищує заданий розмір, було приховано, але, на жаль, на сторінці, що відобразилася у браузері, немає ніяких свідчень чи є ще текст, крім того, що ми бачимо.

Властивість **overflow: auto**; створює полоси прокрутки у тому напрямі, де є текст, що перевищує наданий розмір елементу:



Значення overflow: scroll; у будь-якому випадкові створить вертикальну і горизонтальну полоси прокрутки. Але слід брати до уваги і розміри цих полос при операціях із додавання ширини і висоти елементам.

Створення «гумових» блоків. Адаптивна верстка.

Розглянемо приклад створення div елементів, розміри яких автоматично змінюються при зміні розмірів вікна браузеру. Такий дизайн сторінки називають адаптивною верстkою, або «*гумовим*».

Створимо елемент <body> додав у ньому 5 елементів , для яких вкажімо висоту і додамо видиму рамку:

Фрагмент HTML-коду	**Стиль CSS**

<body>
 	
</body>
	div {
	height: 200px;
	border: 2px dotted blue;
	}

У браузері побачимо п’ять блоків висотою 200px, які розташовані один за одним, і по ширині займають всю ширину вікна браузера.
Розташуємо div елементи у одну лінію, це можна зробити
додав обтікання float: left;
або перетворив блок у блочно-строковий елемент, вказав
display: inline-box;
У відносних одиницях ширина вікна браузеру є 100%, тому div елементам задамо ширину 100/5:

```
width: 20%;
```

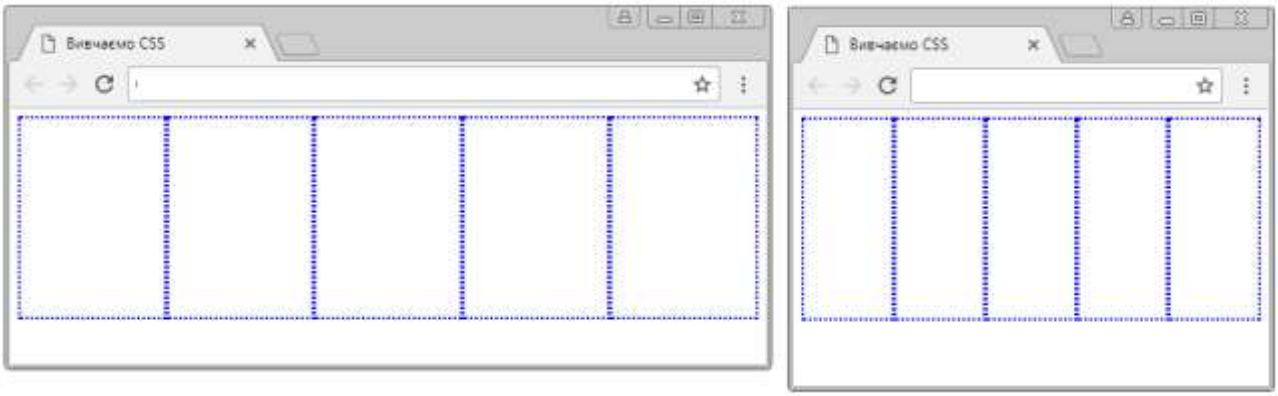
Для того, щоб розміри ширини і висоти включали товщину рамки, тобто були розмірами блоку, а не контенту блока, додамо властивість:

```
box-sizing: border-box;
```

Остаточно css- правила мають вигляд:

```
div {
    height: 200px;
    border: 2px dotted blue;
    float: left;                    /* або display: inline-box; */
    width: 20%;
    box-sizing: border-box;
}
```

І змінюючи розміри вікна браузеру можна бачити ширші чи вужчі елементи:



Зверніть увагу, ми розглянули приклад «гумових» по ширині елементів, висота цих елементів лишається 200px.

За аналогією можна створити «гумові» елементи і за висотою, і одночасно за висотою і шириною. Для цього слід використовувати значення у відсотках.

Приклад, Розробимо дизайн web- сторінки із «гумовою» частиною.

Для цього створимо елемент<body>:

```
<body>
    <div id="header">Шапка сайту</div>
    <div id="left">Лівий вертикальний блок</div>
    <div id="right">Правий вертикальний блок</div>
    <div id="middle">"Гумовий" центральний блок</div>
    <div id="footer">Підвал сайту</div>
</body>
```

Додамо сторінці стиль:

```
#header {
    width: 100%;                /* відносна ширина блоку*/
    margin: 0px;
    text-align: center;
    background-color:#ff9999;
}
#right {
    float: right;               /* обтікання зправа */
    width: 200px;               /* задана ширина блоку  */
    margin: 20px 0px 0px 0px;
    text-align: right;
    background-color:#99ff99;
    height: 400px;
}
#middle {
    margin: 20px 220px 10px 220px;      /* розміри полів  */
    background-color:#9999ff;
```

```
        height: 400px;
        text-align: center;
}
#footer  {
        margin: 0;
        border: solid 1px Dark;
        background-color: #dbc1c1;
        font-size: 10px;
        text-align: center;
        clear:both;        /* скасовується обтікання з обох боків */
}
#left {
        background-color:#fdff5e;
        margin: 20px 0px 0px 0px;
        width: 200px;              /* задана ширина блоку  */
        float:left;                /* обтікання зліва */
        height: 400px;
}
```

І отримаємо у браузері

Основний «гумовий» дизайн зосереджений у правилах, що виділені жирним шрифтом. Решта правил у css стилі прикладу додані «для красоти» ☺.

Тести і завдання для самостійної роботи

Оберіть вірне твердження у наступних питаннях:

Питання 1 **Як розшифрувати абревіатуру CSS?**

1. Creative Style Sheets
2. Colorfull Style Sheets
3. Computer Style Sheets
4. Cascading Style Sheets

Питання 2 **Яка властивість CSS відповідає за висоту букв у тексті?**

1. text-size
2. font-size
3. size
4. text

Питання 3 **У якому HTML елементі слід підключати файл стилів CSS?**

1. <body>
2. <head>
3. <script>
4. наприкінці документу

Питання 4 **Як встановити жирний шрифт для параграфа?**

1. <p style=”font-size: bold;”>
2. <p style=”text-size: bold;”>
3. p { text-size: bold; }
4. p { font-weight: bold; }

Питання 5 **Який HTML атрибут дозволяє записувати стилі in-line?**

1. style
2. styles
3. class
4. align

Питання 6 **Як задати маркери для елементів списку <li> у вигляді «квадратиків»?**

1. list-style-type: square;
2. list -type: square;
3. list: square;
4. just: square;

Питання 7 **Оберіть вірний синтаксис у наступних правилах CSS**

1. { div; color: black; }
2. div: color = black;
3. { div: color = black }
4. div { color: black; }

Питання 8 **Як звернутися до ідентифікатору test?**

1. test
2. #test
3. .test
4. *test

Питання 9 **Як правильно додати коментар у CSS правилах?**

1. <!--коментар-->
2. // коментар
3. /* коментар */
4. // коментар//

Питання 10 **Як звернутися до елементів з класом test?**

1. test
2. #test
3. .test
4. *test

Питання 11 **Як правильно згрупувати елементи?**

1. вказати селектори, розділяючи їх пробілом
2. вказати селектори, розділяючи їх комою «,»
3. вказати селектори, розділяючи їх через слеш «/»
4. записувати кожний селектор з групи з нового рядка

Завдання 1

Створіть html файл, який містить заголовки першого (h1) і другого (h2) рівнів, текстовий абзац з кількох речень. 2-3 слова з абзацу загорніть у елемент span.

Додайте до сторінки CSS стиль, який міститься у окремому файлі my_style.css, і надає сторінці вигляд:

- ✓ заголовок першого рівня (h1)вирівняно за центром, шрифт Courier, розмір 36px, жирного і курсивного накреслення, червоного кольору;
- ✓ заголовок другого рівня (h2) вирівняно ліворуч, шрифт Arial, розмір 24px, курсивного накреслення, зеленого кольору;
- ✓ параграф (p) шрифт Times New Roman, розмір 18 px, рожевого кольору, вирівняно за шириною, міжрядковий інтервал подвійний, відступ («червона строка абзацу») 2см;
- ✓ виділені слова параграфу (span) колір синій, підкреслені, розріджені на 4pt

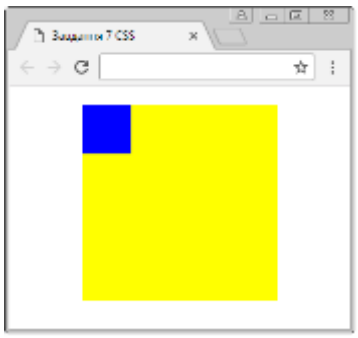
Завдання 2

Створіть html файл, елемент <body> якого такий:

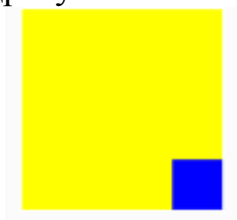
```
<body>
    <div class="parent">
        <div class="child"></div>
    </div>
</body>
```

Підключіть до html файлу файл css-стилю myStyle.css з такими правилами:

```
.parent {
    width: 200px;
    height: 200px;
    background-color: yellow;
    margin: 20px auto;
}
.child {
    width: 50px;
    height: 50px;
    background-color: blue;
}
```

Ви отримаєте 

Відредагуйте css правила таким чином, щоб «синій квадрат» розташувався у правому нижньому куті «жовтого квадрату».

А саме 

Для цього достатньо додати у кожному селекторі css правил, що є, правила позиціювання елементів.

Завдання 3

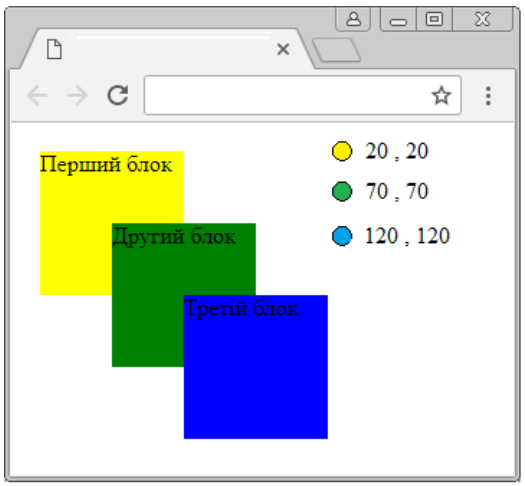
Створіть html файл, елемент <body> якого складається з 5 елементів .
Підключіть до html файлу файл css-стилю myStyle.css з такими правилами:

- Блоки розташовані у лінію, а не один за одним вертикально. Використовуйте правило **display: inline-block** , а ні float: left
- Задайте висоту блоків 150px.
- Укажіть, що розміри блоку включають і розміри (товщину) рамки.
- Задайте рамку товщиною 2px, пунктирну, синього кольору.
- Задайте фон елементів лінійний градієнт з права на ліво від світло-жовтого до зеленого кольору.
- Зробіть так, щоб ваш «макет» div-блоків масштабувався при зміні ширини вікна браузера (був «гумовий» по ширині).

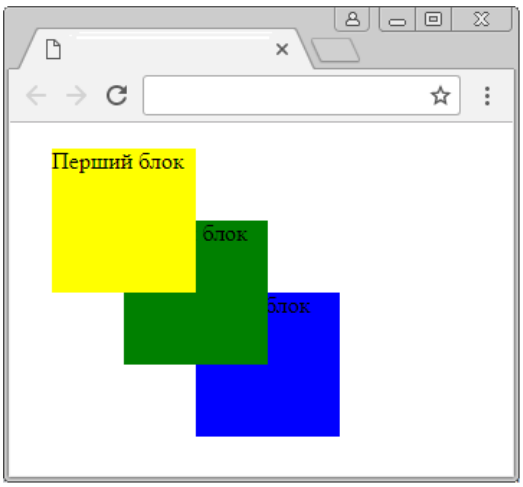
Завдання 4

4.1. Створіть проект, у якому три div- елемента розміром 100×100px, розташовані у абсолютному позиціювання так, що відповідні верхні ліві кути елементів містяться у координатах (див. Мал. 1)

4.2. Розташуйте div-блоки так, як вони виглядають на Мал. 2.
CSS правила розмістіть у елементі <style> html-коду сторінки.
Кожний div- елемент ідентифікуйте і використовуйте ці селектори.
Уникайте дублювання правил, обираючи відповідні селектори.


Перший блок
Другий блок
Третій блок
20 , 20
70 , 70
120 , 120

Мал. 1.


Перший блок
блок
блок

Мал. 2.

Основи JavaScript

Мова програмування JavaScript (у подальшому JS) – це найбільш популярна мова в середовищі Internet, яка є інтерпретованою мовою. Тобто програмні коди, що називаються *скріпти* (сценарії), виконуються без попередньої компіляції.

JS надає Web - сторінкам можливість реагувати на дії користувача (зазвичай, це дії мишею на сторінці), тобто Web – сторінка стає інтерактивною. JS – це мова сценаріїв і має справу з готовими програмними компонентами.

Скріпти JS відпрацьовують у більшості браузерів, таких як Intenet Explorer, Firefox, Chrome, Opera та Safari.

Скріпти зазвичай містяться прямо у HTML коді, але можуть міститися і в окремому файлі (*.js), який приєднується до сторінки.

Для використання JS не потрібна покупка ліцензії.

Важливі моменти синтаксису мови JS:

- ✓ всі ідентифікатори є регістрозалежними;
  наприклад, **MyId** та **myId** це різні ідентифікатори
- ✓ кожна строчка JS – це інструкція для браузера. Браузер читає і виконує код строчка за строчкою;
- ✓ кожна інструкція завершується символом « **;** » Цей символ зазначає, що інструкція завершена, і після неї йде наступна інструкція. Рекомендується кожну інструкцію у коді писати з нової строчки;
- ✓ при відображенні Web – сторінки, яка містить JS код, браузер, зустрівши код, опрацьовує його, а потім продовжує завантаження решти контенту сторінки. Цей факт слід брати до уваги обираючи місце розташування JS коду на сторінці;
- ✓ у JS коді можна додавати коментарі. Слід розрізняти коментарі, які записані у ОДНУ строку, чи у ДЕКІЛЬКА строк, бо такі коментарі мають різний синтаксис.

Для написання JS коду будемо використовувати професійний off-line додаток Sublime Text 3 з браузером Google Chrome. Додаток Sublime Text 3 можна завантажити з сайту розробника https://www.sublimetext.com/3 , обрав відповідну операційну систему.

За наявності постійного підключення до мережі Інтернет деякі приклади можна розглядати у on-line редакторові http://liveweave.com/ , але беріть до уваги, що написані коди відразу й виконуються і не зберігаються.

Розташування JS коду на сторінці:

Створимо у редакторі Sublime Text 3 стандартний HTML код:

```
<!DOCTYPE html>
<html>
<head>
        <title>Розміщуємо JS</title>
        <meta charset="utf-8">
</head>
<body>
        <p>Параграф на сторінці</p>
</body>
</html>
```

Код JS , *наприклад,*

```
var a="Привіт! Це працює JS";
alert(a);
```

слід додавати у тег <script type="text/javascript"></script>.

Розглянемо варіанти розміщення JS коду. Уважно стежте за відображенням сторінки (її контенту, що містить параграф) у браузері.

Порядок відпрацювання скріпта може залежати і від налаштувань браузера.

Варіант 1 **(скріпт розташований у тегі <head>)**	**Варіант 2** **(скріпт розташований у тегі <body> ПЕРЕД контентом).**
<!DOCTYPE html> <html> <head> <title>Розміщуємо JS</title> <meta charset="utf-8"> <script type="text/javascript"> var a="Привіт! Це працює JS"; alert(a); </script> </head> <body> Параграф на сторінці </body> </html>	<!DOCTYPE html> <html> <head> <title>Розміщуємо JS</title> <meta charset="utf-8"> </head> <body> <script type="text/javascript"> var a="Привіт! Це працює JS"; alert(a); </script> Параграф на сторінці </body> </html>
Контент сторінки буде відображений ПІСЛЯ того, як відпрацює скріпт	

Тому зазвичай JS код розміщують перед закриваючим тегом </body>:

Варіант 3 (скріпт розташований наприкінці контенту)	**Варіант 4 (скріпт розташований у окремому файлі).**
<!DOCTYPE html> <html> <head> <title>Розміщуємо JS</title> <meta charset="utf-8"> </head> <body> Параграф на сторінці <script type="text/javascript"> var a="Привіт! Це працює JS"; alert(a); </script> </body> </html>	<!DOCTYPE html> <html> <head> <title>Розміщуємо JS</title> <meta charset="utf-8"> </head> <body> Параграф на сторінці <script type="text/javascript" src="JS/MyScript.js"> </script> </body> </html>
Спочатку буде відображена сторінка, а потім почне працювати скріпт	

Нижче наведений зміст файлу скріпта із одно строковим і багато строковим коментарями

Первый скрипт.js

```
1  var a="Привіт! Це працює JS"; // оголошення і присвоювання для змінної a
2  /* Вікно повідомлення
3  Після того, як користувач ознайомився із повідомленням,
4  слід клікнути ОК, вікно закриється */
5  alert(a);
```

Змінні і типи

Правила іменування змінних:

Мовою спілкування між розробниками програмного забезпечення визначена англійська. Зовнішній вигляд коду має бути «читабельним», зрозумілим розробникові у будь-якій країні. Розробники з різних країн стикаються із необхідністю доопрацьовувати коди інших розробників. При написанні коду прийнято:

- ✓ не використовувати транслiт, підбирати назви зі словами англійської мови;
- ✓ короткі назви (1-2 літери) надаються, зазвичай, тільки індексам;
- ✓ змінні, ім'я яких мстить декілька слів, надаються за правилом camelCase – перше слово пишеться із маленької літери, наступні – з великої (*наприклад*, myMessage newFoodProducts);
- ✓ ім'я змінної відповідає її вмісту;
- ✓ у якості імен не використовуйте зарезервовані слова (*такі як*, if case class тощо)

НЕ ЗАБУВАЙТЕ ПРО РЕГІСТРОЗАЛЕЖНІСТЬ ІМЕН!

Ім'я змінної носить назву ***ідентифікатор змінної***.
Перед використання змінну треба оголосити (для цього використовується var), а потім використовувати:

```
var x;
x=5;
```

А можна оголосити і використати в одній дії

```
var myMessage='Привіт!'
```

Примітиви

number числова змінна

```
var number=123;
var anotherNumber=123.456;
```

string строкова (текстова) змінна

```
var str=''Текстовая строка'' або так var str='Текстовая строка'
```

у JS одинарні (') і подвійні ('') лапки рівнозначні.

boolean булевська (логічна) змінна

```
var checked=false;
checked=true;
```

Спеціальні значення:

null

```
var age=null;        // змінна оголошена, але її значення не визначене
var x;               // змінна x має значення null
```

undefined

Значення, що повертається, якщо змінна є null

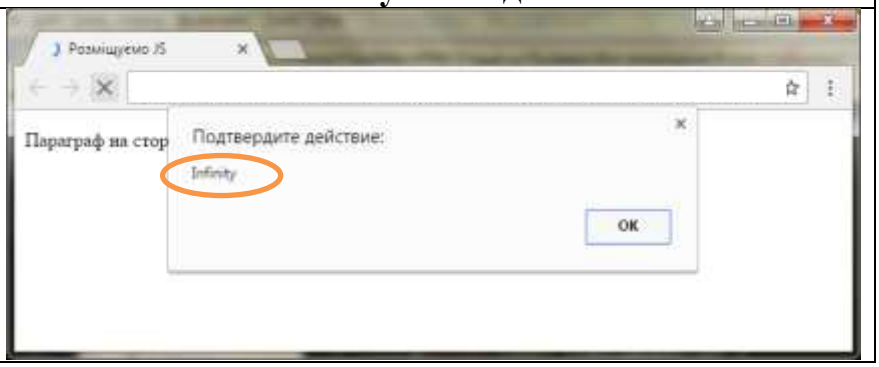
Наприклад, результатом дії наступного JS коду буде вікно повідомлення:

JS код	Результат дії
<!DOCTYPE html> <html> <head> <title>Розміщуємо JS</title> <meta charset="utf-8"> </head> <body> Параграф на сторінці <script type="text/javascript"> var x; alert(x); </script> </body> </html>	

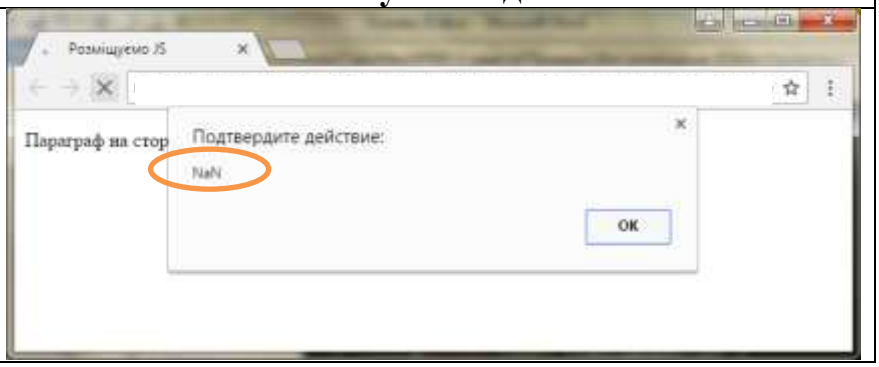
Значення, що повертаються в результаті дії:

infinity нескінченність

JS код	Результат дії

<script type="text/javascript">
var x=0;
alert(5/x);
</script>
	

NaN зазвичай, отримуємо в результаті помилкової математичної операції

JS код	Результат дії
<script type="text/javascript"> var x='notNumber'; alert(5/x); </script>	

Складні типи

array масиви із числовими індексами.

Масиви дають можливість зберігати і використовувати декілька значень під одним ім'ям. Кожне значення у масиві має свій порядковий номер (індекс). Індексація (нумерація) масивів завжди починається з 0 (нуля). Данні у масиві можуть бути одного типу, а можуть бути і різних типів (такі масиви називаються ***змішаними***).

Оголосити масив var array=[]; // використовуються квадратні дужки

Числовий масив var arrX=[1,12,3.17]; // розділяємо значення комою

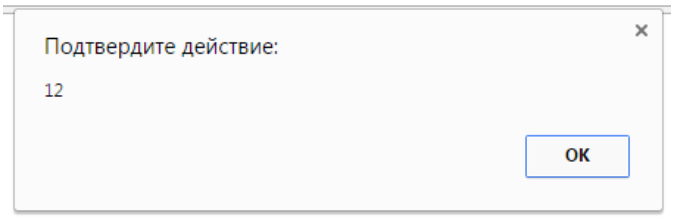
Текстовий масив var arrStr=['банан','лимон','апельсин','ківі'];

Змішаний масив var arrMix=[5.28,'банан',true];

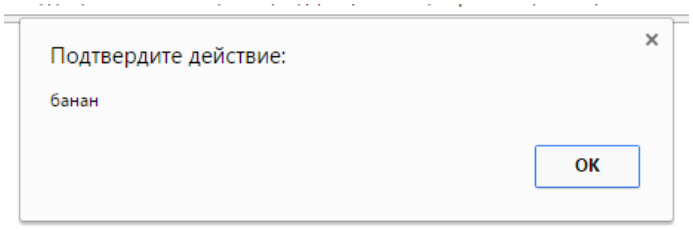
Звернутися до конкретного значення у масиві можна за допомогою його індексу.

Наприклад, для наведених вище масивів

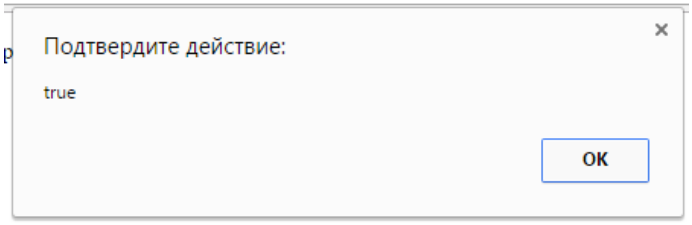
alert(arrX[1]);



alert(arrStr[0]);



alert(arrMix[2]);



object об’єкт.
За допомогою об’єктів будують досить складні конструкції.
Оголосити об’єкт var objectPerson={}; // використовують фігурні дужки.

Структура об’єкту – це запис властивостей і відповідних значень. Властивості розділяються комою (**,**), між назвою властивості і її значення ставиться двокрапка (**:**)

```
var object={property: value};
            властивість  значення
```

Наприклад,

```
var user={
        name: 'Василь',
        surname: 'Петренко'};
```

Об’єкти можуть вміщувати інші об’єкти, *наприклад*,

```
var obj={
        name: 'Василь',
        surname: 'Петренко',
        job:{
                first='Логістик',
                second='Брокер'
        }
};
```

Модальні вікна для діалогу з користувачем

Надати можливість користувачу отримати повідомлення під час завантаження Web- сторінки у браузері чи отримати від користувача дані (примітиви) можна за допомогою команд, що формують відповідні *модальні вікна*. Модальне вікно припиняє дію браузера, доки користувач не дасть відповідь за допомогою кліка миші на відповідній кнопці у модальному вікні.

Надати дані у повідомленні користувачу alert

Як ми бачили у попередніх прикладах, повідомлення користувачу можна надати за допомогою вікна

alert('Повідомлення');

Під час виконання команди alert браузер припинить наступні дії JS коду до тих пір, поки користувач на клікне мишею кнопку 

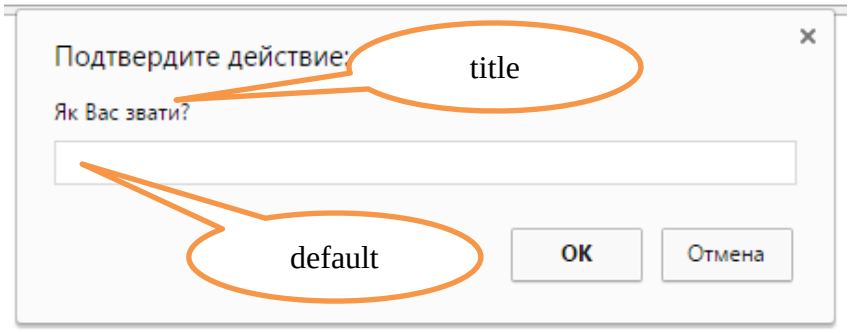
у модальному вікні.

Отримати дані від користувача prompt

var result=prompt(title,default);

Наприклад,

```
var name=prompt('Як Вас звати?', '');
```

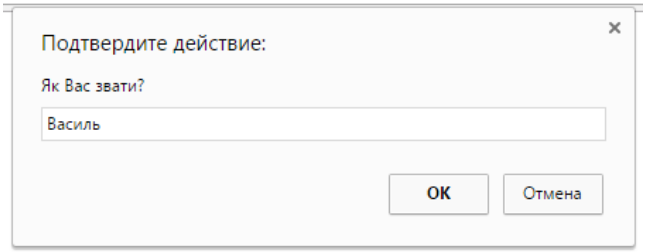


Користувач вводить дані (тут своє ім'я) у полі і клікає ОК.
Змінна name стане дорівнювати введеним даним. Якщо користувач клікне 

, то змінна name отримає значення null і стане undefined.

Для появи вікна prompt із вже заповненим полем, треба написати у коді, *приміром*

```
var name=prompt('Як Вас звати?','Василь');
```



Тоді достатньо тільки клікнути ОК, і nane отримає дані 'Василь'. Але за потребою можна замість 'Василь' набрати у полі вікна будь-які інші дані.

Наведений нижче JS код дає можливість поспілкуватися із комп’ютером ☺:

```
var name=prompt('Як Вас звати?', '');
alert('Вас звати  ' + name);
```

Зауваження. Слід зазначити, що змінна, яка отримала дані з вікна prompt буде мати тип string. Якщо необхідно, щоб дані були числом перед prompt треба поставити символ плюс «+».

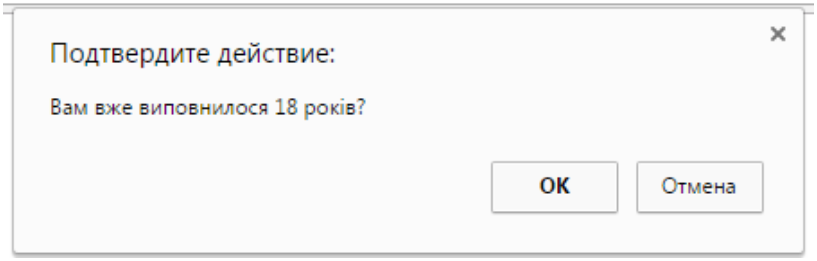
```
var myValue=+prompt(‘Введіть число’,’’);
```

Отримання від користувача відповіді ТАК чи НІ на питання confirm

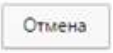
var result=confirm(question);

Наприклад,

```
var isAdult=confirm('Вам вже виповнилося 18 років?');
```



Користувач клікне  - змінна isAdult стане true.

Користувач клікне  - змінна isAdult стане false/

Дії з масивами

У класичному програмуванні оперують із такими поняттями, як *черга* і *стек*. Черга і стек – це послідовності елементів з певними правилами доступу до поповнення і обробки елементів.

Черга – це така впорядкована послідовність елементів, до якої нові елементи можуть додаватися тільки у кінець, а обробляється така послідовність із початку (**FIFO**, First In — First Out, «першим прийшов – першим пішов»)

Стек - це така впорядкована послідовність елементів, до якої нові елементи додаються у кінець і обробляються, починаючи із кінця
(**LIFO**, Last In— First Out, «останнім прийшов – першим пішов»)

Масиви можуть оброблятися за технологією черги чи стеку.

Видалення і додавання елементів у масив. Методи pop/push, shift/unshift.

Методи записують після імені масиву через крапку «**.**». У круглих дужках надають параметри методу, якщо вони є, але кругли дужки обов'язкові.:

масив.метод();

pop – видалення останнього за номером елементу у масиві. Відповідно змінюється і довжина масиву.

Наприклад,

```
var fruits = ["Банан", "Апельсин", "Слива"];                //масив з трьох елементів
fruits.pop() ;                          // видалили "Слива"
console. log (fruits );                 // Банан, Апельсин    масив з двох елементів
```

push – додати один або кілька нових елементів у кінець масиву. У круглих дужках через кому надаються елементи, які будуть додані у масив. Відповідно зміниться і довжина масиву.

Наприклад,

```
var fruits = ["Банан", "Апельсин", "Слива"];
fruits.push("Яблуко","Груша","Смородина") ;              // додали три елемента
console.log (fruits) ;        // Банан, Апельсин, Слива, Яблуко, Груша, Смородина
```

shift – видалити перший елемент масиву. Відповідно змінюється і довжина масиву.

Наприклад,

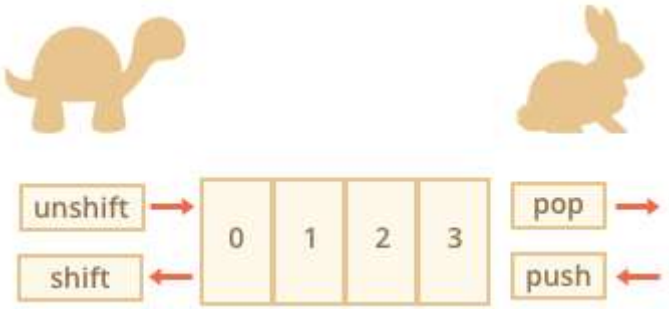
```
var fruits = ["Банан", "Апельсин"];
fruits.shift() ;                        // видалити перший елемент
console.log (fruits) ;                  // Апельсин
```

unshift – додати один або кілька елементів на початок масиву. Елементи, що будуть додаватися, перелічені у круглих дужках через кому. Відповідно зміниться і довжина масиву.

Наприклад,

```
var num = [5, 7];
num.unshift(1,3) ;          // додати два елемента
console.log (num) ;                 // 1, 3, 5, 7
```

Примітка. Виконання розглянутих методів впливає на швидкодію виконання скріпту.
Методи pop/push виконуються швидко, а методи shift/unshift повільно.



Тобто працювати із кінцем масиву швидше, ніж із його початком.
Чим більше елементів у масиві, тим повільніше виконуються методи.

Перетворення масива у строку. Метод join

Метод **join** повертає елементи масиву у вигляді строки. У круглих дужках можна вказати символ, яким будуть розділені елементи масива у строці. За замовчування розділителем є кома «**,**».

Приклад 1.

```
var num = [4.5, 2,8];
var phrase=num.join() ;          // розділитель не вказаний (за замовчуванням)
console.log (phrase) ;                  //  4.5,2,8
```

Приклад 2

```
var words = ['ми', 'вивчаємо','JavaScript'];
var phrase=words.join(' ') ;          // розділити пробілом
console.log (phrase) ;                  //  ми вивчаємо JavaScript
```

Зміна порядку і сортування елементів масиву. Методи reverse і sort.

reverse – повертання масиву із елементами у в зворотньому напрямку

Наприклад,

```
var num = [4.5,2,8,34];
console.log(num);          // 4.5, 2, 8, 34
num.reverse();
console.log(num); // 34, 8, 2, 4.5
```

sort – сортування за алфавітом. ***Зверніть увагу***, що числовий масив метод sort сортує, як строкові елементи.

Приклад 1.

```
var fruits = ["Банан", "Апельсин", "Слива"];
fruits.sort();
console.log(fruits);  // Апельсин, Банан, Слива
```

Приклад 2 (для числових масивів).

var num = [4.5,2,8,3];
num.sort();
console.log(num); // **2, 3, 4.5, 8**
	var num = [4.5,25,8,34];
	num.sort();
	console.log(num); // **25, 34, 4.5, 8**

Для сортування саме чисел слід додати функцію у параметри методу sort:

```
num.sort(function(a,b) { return a-b; });
```

Функція повертає різницю між двома числами (сусідніми). Якщо різниця менше за нуль (негативна), то пара a b залишається у своєму порядку. Якщо різниця більше нуля (позитивна), то пара a b змінює порядок на b a

```
var num = [4.5,25,8,34];
num.sort(
        function(a,b) {
                return a-b;
        }
);
console.log(num);                    // 4.5, 8, 25, 34
```

Витяг елементів масиву. Метод slice

slice(begin,end) – наприклад **slice(2,5)** – витягує і повертає в інший масив елементи з індексами 2, 3, 4.
Зверніть увагу, з елемента за номером begin ДО елементу за номером (end – 1) включно, а елемент за номером end не входить до елементів, що повертаються.

Наприклад,

```
var fruits=['Банан', 'Апельсин', 'Слива', 'Яблуко', 'Груша', 'Смородина'];
var myChoiceFruits=fruits.slice(1,4);
console.log(myChoiceFruits);  // Апельсин, Слива, Яблуко
```

Додавання елементів у середину масиву із можливістю видалення елементів. Метод splice

splice – надає можливість додавати елементи у масив, починаючи із вказаного індексу. Також надає можливість зазначити скільки елементів видалити, починаючи із вказаного індексу

```
array. splice(begin index, how many to delete, new elements);
                fruits.splice(3,0,’Лимон’,’Ківі’);
```

Перший параметр, тут 3, вказує з якого індексу почнеться додавання.
Такий запис додасть у масив fruits елемент Лимон з індексом 3 і елемент Ківі з індексом 4.
Другий індекс, тут 0, вказує скільки елементів у масиві треба видалити.

Приклад 1 (вставка елементів)

```
var fruits=['Банан', 'Апельсин', 'Слива', 'Яблуко','Груша'];
console.log(fruits);
fruits.splice(3,0,'Лимон','Ківі');
console.log(fruits);
```

```
▶ (5) ["Банан", "Апельсин", "Слива", "Яблуко", "Груша"]
▶ (7) ["Банан", "Апельсин", "Слива", "Лимон", "Ківі", "Яблуко", "Груша"]
```

Приклад 2 (заміна елементів)

```
var fruits=['Банан', 'Апельсин', 'Слива', 'Яблуко','Груша'];
console.log(fruits);
fruits.splice(1,3,'Лимон','Ківі');
console.log(fruits);
```

```
▶ (5) ["Банан", "Апельсин", "Слива", "Яблуко", "Груша"]
▶ (4) ["Банан", "Лимон", "Ківі", "Груша"]
```

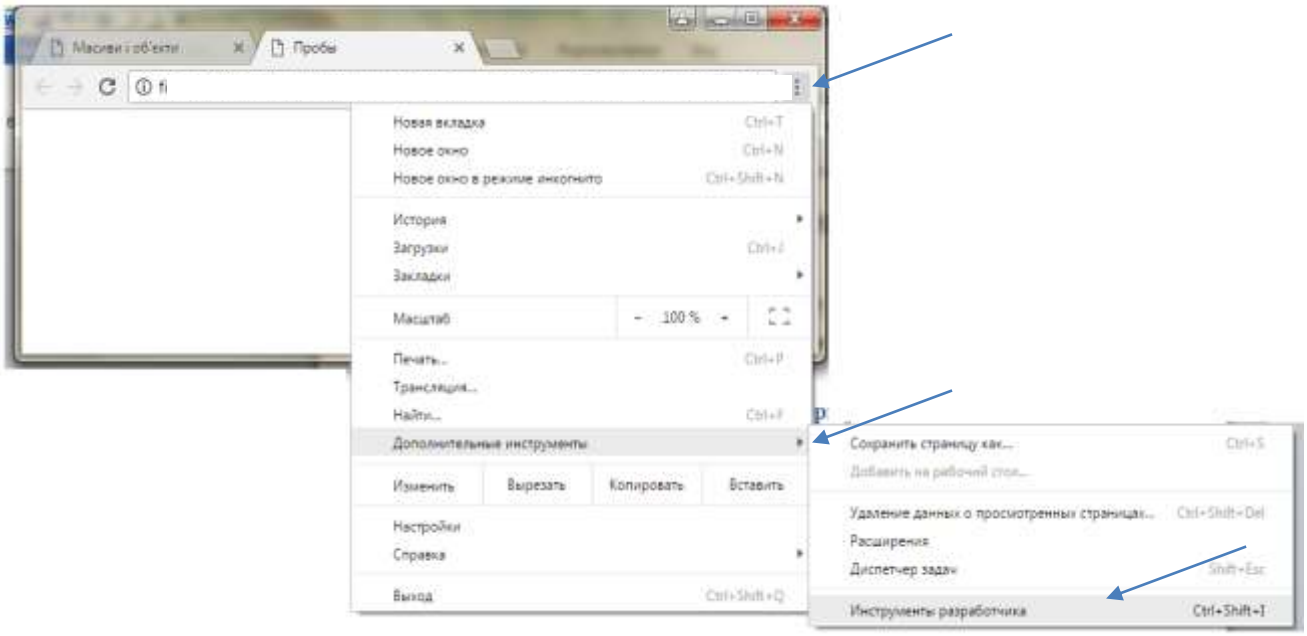
Приклад 3 (тільки видалення елементів)

```
var fruits=['Банан', 'Апельсин', 'Слива', 'Яблуко','Груша'];
console.log(fruits);
fruits.splice(2,2);   // видалення елементів з індексами 2 і 3
console.log(fruits);
```

```
▶ (5) ["Банан", "Апельсин", "Слива", "Яблуко", "Груша"]
▶ (3) ["Банан", "Апельсин", "Груша"]
```

У наведених прикладах результат виводився у **консоль браузера** за допомогою оператора console.log()

Подивитися цей інструмент можна натиснув функціональну клавішу F12, або обрав у меню



Оператори

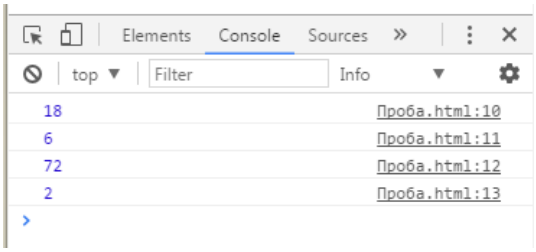
Як і у попередньому розділі для проведення вправ по ознайомленню із результатами дії операторів будемо використовувати виведення результатів у консоль браузера (console.log()).

Арифметичні оператори

Символи **+ - * /** визначають «добре відомі» операції. Перевірте їх дію у наступному *прикладі*:

```
var a=12;
var b=6;
        console.log(a+b);
        console.log(a-b);
        console.log(a*b);
        console.log(a/b);
```

У консолі отримаємо



Зауваження. Символ плюс «+» використовується не тільки для суми чисел, а і для *конкатенації* (склеювання) строкових змінних (констант).
Якщо в операції « +» хоча би один з операндів буде типу string, то і результат операції буде типу . string

Порівняйте результати таких кодів:

Обидва операнда типу number	***Один з операндів типу string***

var a=12;
var b=6;
console.log(a+b);
	var a=12;
	var notNumber='6';
	console.log(a+ notNumber);

Інкремент (додати 1) і декремент (відняти 1)

У JS досить часто доводиться збільшувати (або зменшувати) значення змінної на одиницю (1).
Традиційним оператором у більшості мов програмування є такий, *наприклад*

```
var j=1;
j=j+1;
```

Тобто спочатку змінна j була визначена із значенням 1,
а потім нове значення j стало 1+1 («нове» = «те, що було» +1)
(*ітераційний оператор*).

У JS можна скоротити записи і замість оператора присвоювання (j=j+1) записати **інкремент** (j++):

```
var j=1;
j++; // нове значення j стало дорівнювати 2
```

За аналогією діє і **декремент** (k--) замість оператора присвоювання (k=k-1)

```
var k=3;
k--;  // «два мінуси підряд» і нове значення k стало дорівнювати 2
```

Увага! Оператори інкремент і декремент застосовуються тільки для змінних, їх не можна використовувати поряд із константами (**невірно** писати 5++ чи 6--)

Оператори порівняння

Результатом застосування операторів буде булевська змінна (true або false).

<	менше
>	більше
<=	не більше
>=	не менше
==	чи дорівнює значення
===	чи дорівнює значення і тип (суворе порівняння)

Подивіться, якими будуть результати :

```
console.log('01'==1); // результатом буде true
```

Тут константа типу string (01) була визнана такою, що дорівнює числу 1.

```
console.log('01'===1); // результатом буде false
```

При суворому порівнянні «текст» і «число» були визнані такими, що не дорівнюють одне одному.

!= не дорівнює по значенню

!== не дорівнює по значенню і типу (суворо не дорівнює)

Подивіться, якими будуть результати :

```
console.log('01'!=1); // результатом буде false
```

Як ми вже бачили при несуворому порівнянні «текст» 01 и «число» 1 визнаються такими, що дорівнюють одне одному.
Тому і твердження «01 не дорівнює 1» було визначене, як хибне (false).

```
console.log('01'!==1); // результатом буде true
```

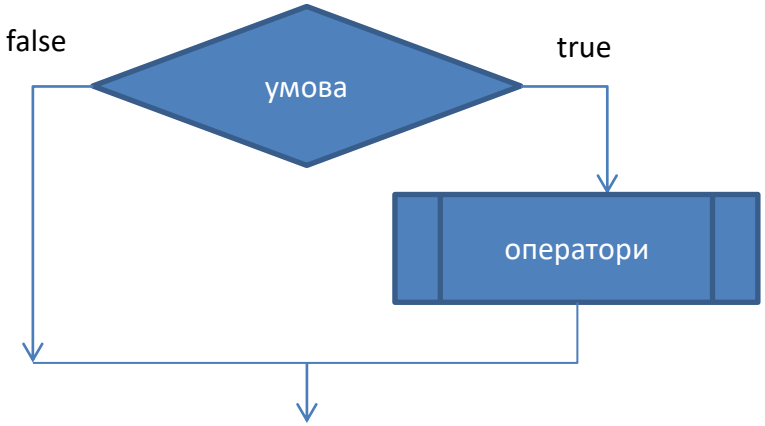
А ось при суворому порівнянні 01 і 1 не дорівнюють одне одному, тому твердження «01 не дорівнює 1» визнане вірним (true).

Логічна операція		*Приклад*
and	&&	x>2 && y<3
or	||	x<0 || y<0
not	!	!(x > 1 && x < 10)

Умовний оператор
оператори if , else if і короткий запис оператора «?»

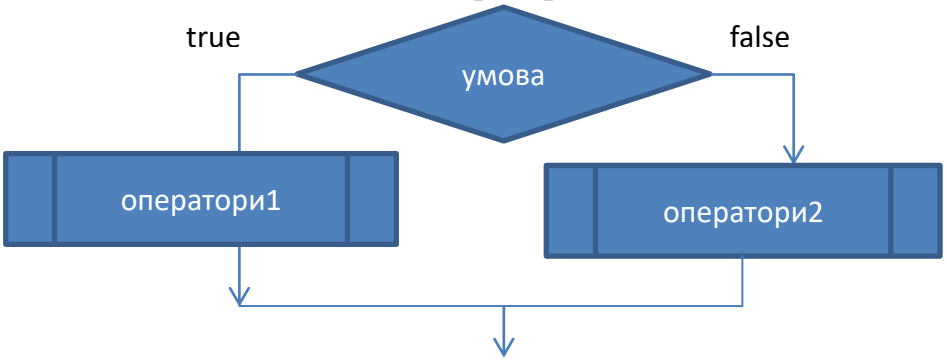
Коли виникає необхідність виконувати ті чи інші дії, в залежності від певної умови, використовується оператор if.

- Треба виконувати дії (оператори) тільки тоді, коли умова (логічне твердження) є істинною:




```
if («умова») {
«оператор»;
}
```

- Якщо умова істинна, то треба виконувати «оператори 1», а якщо хибна, то «оператори 2»:



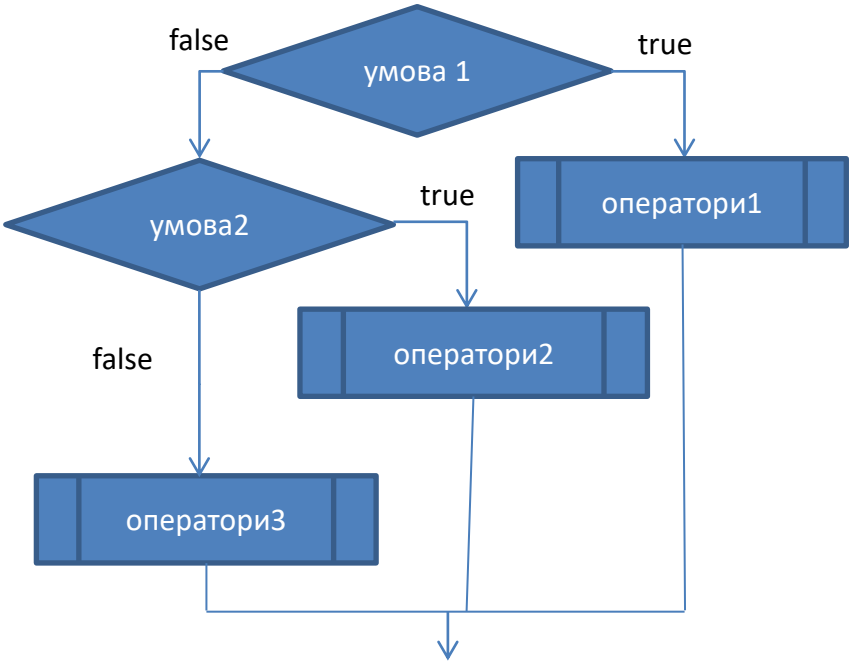

```
if («умова») {
«оператор1»;
} else {
«оператор2»;
}
```

Приклад 1

```
var year=+prompt('Укажіть рік появи мови JS','');
if (year===1995) {
alert('Ви відповіли вірно');
} else {
alert('Ви помилилися');
};
```

Користувачу пропонується ввести рік створення мови JS, якщо він введе 1995, то отримає повідомлення «'Ви відповіли вірно'». При будь-яких інших введених даних користувач отримає повідомлення «'Ви помилилися'».

- Перевірка кількох умов:




```
if (умова1) {
операторі1;
}else if (умова2) {
операторі2;
}else {
операторі3;
}
```

Умова в операторі if завжди перетворюється до булевого типу (true або false). Такі примітиви, як «число 0», пуста строка «’’», null, undefined і NaN завжди є false, решта – true.

Приклад 2 (оператор else if)

```
var year=+prompt('Укажіть рік появи мови JS','');
if (year===1995) {
alert('Правильно!');
} else if(1990<year && year<2000){
alert('Hi, але близько');
} else {
        alert('Зовсім не вгадали');
};
```

Порівняно із попереднім завдання (див. *Приклад* 1), користувачу «надається підказка», якщо введене ним число було з інтервалу (1990; 2000).

Приклад 3 (короткий запис «?»)

Користувач вводе свій вік. Якщо він повнолітній, то доступ отримано, інакше доступ закритий.

«**?**» (знак запитання) – єдиний оператор, у якого є аж три аргументи, в той час, коли звичайні оператори мають один чи два аргументи. Тому оператор «?» називають *тернарним оператором*.
Оператори з одним аргументом називаються *унарними*, а з двома – *бінарними*.

Розгорнутий запис (if)	Короткий запис (?)
var age=+prompt('Скільки Вам років?',''); var access; **if (age>18) {** **access=true;** **} else {** **access=false;** **};** alert(access);	var age=+prompt('Скільки Вам років?',''); var access; **access=age>18 ? true:false;** alert(access);

Конструкція switch

Умовний оператор (if)	Оператор (switch)
var value=+prompt('Скільки буде 2+2 ?',''); **if(value===3){** **alert('Не вистачило');** **} else if(value===4){** **alert('Правильно!');** **}else if(value===5){** **alert('Перебор');** **}else {** **alert('Зовсім погано');** **};**	var value=+prompt('Скільки буде 2+2 ?',''); **switch(value){** **case 3:** **alert('Не вистачило');** **break;** **case 4:** **alert('Правильно!');** **break;** **case 5:** **alert('Перебор');** **break;** **default:** **alert('Зовсім погано');** **}**;

Увага! Якщо у конструкції switch прибрати команду break (переривання), то продовжать виконуватися наступні case'и один за другим. Таким чином, директива break запобігає тому, щоб switch виконувався далі, доки не зустрінеться default.
default означає «всі інші значення змінної, що не увійшли до наявних case'ів.
Декілька значень case можна групувати, для цього вони записуються просто один під одним. Групування дозволяє отримати інтервальні значення для case' ів (*наприклад*, від 2 до 3, слід згрупувати case 2: і case 3:) .

Повторення дій (цикли)

Цикли – це оператори, за допомогою яких реалізуються алгоритми повторення дії. У JS є чотири типа операторів циклу:

Умовні, коли кількістю повторів керує умова. Повтори здійснюються, доти умова є істиною (true), і кількість повторів заздалегідь невідома:

Цикл while (цикл з передумовою)

Загальна структура оператора	*Приклад*	

while (умова) {
….тіло циклу;
}
	var i=0;
	while (i<3) {
	alert(i);
	i++;
	}
		Початкове значення і дорівнює 0.
		Перевіряється умова i<3 на значення true
		У «вікні повідомлень» виводиться і
		і збільшується на 1 і знов перевіряється умова і, якщо умова залишається true, дії «тіла циклу» повторюються

Цикл do while (цикл з післяумовою)

Загальна структура оператора	*Приклад*	

do {
….тіло циклу;
} while (умова);
	var i=0;
	do {
	alert(i);
	i++;
	} while (i<3);
		Початкове значення і дорівнює 0.
		У «вікні повідомлень» виводиться і
		і збільшується на 1
		Перевіряється умова i<3 на значення true
		І дії «тіла циклу» повторюються

Увага! При реалізації умовних циклів треба добре обдумати алгоритм, у якому обов'язково передбачено зміну умови у тілі циклу. В іншому випадку цикл може «здійснюватися нескінченно»

Цикли ***із лічильником*** – кількість повторів циклу відома заздалегідь, тобто повтори можна порахувати:

Цикл for (відома кількість повторів)

Загальна структура оператора	*Приклад*

for (begin;end;step) {
….тіло циклу;
}
	for (var i=0; i<3; i++) {
	alert(i);
	}

Примітка. Будь-яка з частин begin;end;step оператору for може бути вилучена, але повинно зберегтися «місце» частини.

Порівняйте такий запис скріпта до попереднього прикладу:

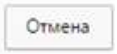
```
var i=0
for ( ; i<3; i++) {
alert(i);
}
```

Переривання у циклі. Оператори break и continue

За допомогою оператора break можна здійснити вихід з циклу до його завершення.

Розглянемо приклади.

Управління циклом діями користувача на web-сторінці:

1. Користувачу надається можливість вводити число до тих пір поки він не клікне кнопку 


```
while(true) {
        var value=prompt('Задайте число','');
        if (value==null) {
        break;
        }
        alert('Ви ввели ' + value);
}
```

2. Пропустити ітерацію (не виконувати дії циклу для ітерації за вказаним номером)

```
for (var i=0;i<4;i++) {
        if (i==2){
                continue;
        }
        alert('Виконано ітерацію ' + i);
}
```

Ітерація при i=2 буде пропущена

Цикл для кожного елементу *(*)*

Перебір ***елементів у масиві***:

Задамо масив fruits і виведемо у «вікно» кожен його елемент

```
var fruits=['Яблуко','Банан','Слива','Вишня'];
for(var i=0;i<fruits.length;i++) {
        alert(fruits[i]);
}
```

Тут до масиву fruits застосована властивість length (fruits.length), яка обчислить довжину масиву.

Перебір ***параметрів об'єкту***:
Задамо об'єкт person і виведемо у «вікно» кожен його параметр

```
var person={
        firstName: 'Віктор',
        lastName: 'Костенко',
        age: 23
};
for(var key in person) {
        alert(person[key]);
}
```

Увага!. Цикл for … in має багато запобіжників для використання. Він не рекомендований для масивів,його не можна застосовувати для DOM елементів тощо.

(*) – приклади цього підрозділу надані для загального ознайомлення із синтаксисом оператора «для кожного елементу». При реальному програмуванні ці приклади не є коректними.

Функції. Область видимості змінних.

Функції є центральним компонентом програмування на JS. Функції – це «будівельні блоки» коду, головна мета яких запобігти дублювання частин коду. Функція визначається один раз, а потім багаторазово використовується (говорять, *функція визивається* чи *звернення до функції*).
Приклади вбудованих функцій ви вже бачили на прикладі модальних вікон. Подальшим розвитком функцій є використання *бібліотеки jQuery*, про яку ми поговоримо згодом.
Користувач може створювати і свої функції.

Загальний вид при створенні (**визначенні**) функції:

```
function nameFunction (params,separated,by,conas) {
…..тіло функції;
}
```

Ключове слово **function** , після якого пробіл і ім'я функції. У круглих дужках надається список параметрів. Список може бути й пустим, але круглі дужки обов'язкові.
Параметри функції називають *формальні змінні функції*.

Звертання (визов) до функції здійснюють так:

```
nameFunction (params,separated,by,conas);
```

При звертанні у якості параметрів вказують змінні, які будуть використані функцією саме у цей момент. Такі параметри називають *фактичними змінними функції*.
Звертатися до функції можна і ДО, і ПІСЛЯ її визначення. Тому більш прийнятно всі визначення функції розташовувати наприкінці коду, тобто звертання до функцій у такому коді завжди будуть ДО. Така структура JS коду робить його більш «читабельним».
Розглянемо приклади.

Приклад 1.
Визначення функції без параметрів.

```
function count(){
        var num1=1;
        var num2=2;
        var num3=3;
        var sum=num1+num2+num3;
        console.log(sum);          // результат sum виводиться у консоль браузера
}
```

Зверніть увагу! Якщо ми відобразимо у браузері html- сторінку з цим кодом, то ніяких даних у консолі не з'явиться. Функція тільки визначена, але не виконується браузером при відображенні сторінки.

Для того, щоб функція виконалася, до неї треба звернутися.
Звернемося до нашої функції
count();
і при відображенні сторінки у консолі браузера побачмо число 6.

Приклад 2.

```
function count(num2){
        var num1=1;
        var num3=3;
        var sum=num1+num2+num3;
        console.log(sum); // результат sum виводиться у консоль браузера
}
count(2);
```

num2 формальна змінна

Тут змінна num2 отримала фактичне значення 2

Зверніть увагу! Формальна змінна num2 не визначається за допомогою var при визначенні функції.
А ось у подальшому коді можна записати і так:

```
var fact=4;
count(fact);
```

У консолі отримаємо 8. Значення фактичної змінної fact було передано в функцію count і стало значенням формальної змінної num2.

Приклад 3.
Можна винести у формальні параметри всі змінні, тоді:

```
function count(num1,num2,num3){
        var sum=num1+num2+num3;
        console.log(sum);
}
// Звертання може виглядати так
var a=3;
var b=4;
var c=5.8;
count(a,b,c);
```

І у консолі побачимо 12.8.

Дефолтні (за умовчанням) значення параметрів функції.

Можна зробити таку функцію, у якій будуть надані значення для формальних параметрів навіть тоді, коли користувач при звертанні до функції не вказав їх у переліку фактичних параметрів.

Наприклад.

```
function count(num1,num2,num3){
        var num1=num1 || 1;
        var num2=num2 || 2;
        var num3=num3 || 3;
        var sum=num1+num2+num3;
        console.log(sum);
}
count(3,4);
```

Результатом дії такого коду буде 10.
Тут при визначенні формальних змінних у функції використано знак « || », що інтерпретується, як або (див. Логічні оператори). Тобто дефолтними значеннями змінних num1, num2, num3 будуть числа 1, 2, 3 відповідно, якщо при зверненні змінна буде пропущена ***Увага!*** «Пропускати» можна тільки змінні, які стоять наприкінці списку, або «пропустити» можна весь список. *Наприклад*, помилкою буде запис count(,,4);.

Порівняйте результати визначеної у прикладі функції при таких зверненнях до неї:

```
count(4);        count();
```

Ми описали один із засобів визначення функції (function declaration), як найбільш зрозумілий новачку ☺. Але це не єдиний можливий запис.

Засоби визначення функцій

У попередніх прикладах ми розглянули один із засобів визначення функції, так званий
function declaration

Загалом, засобів 4:

- function declaration
- function expression
- визначення і звертання у одній конструкції
- рекурсія. Звертання до функції міститься у тілі самої функції.

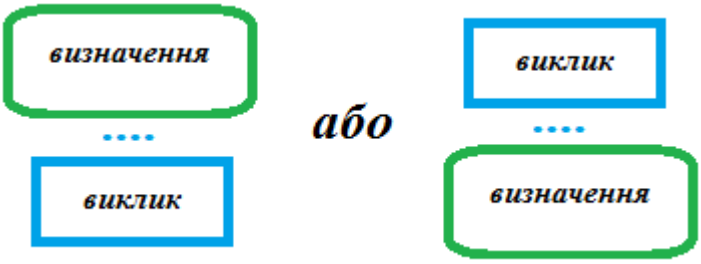
Для **function declaration**:

Структура визначення:

function nameFunction («формальні параметри») {
…..тіло функції;
}

Виклик функції:

nameFunction («фактичні параметри»);



Розташування у JS коді:

Для **function expression**:

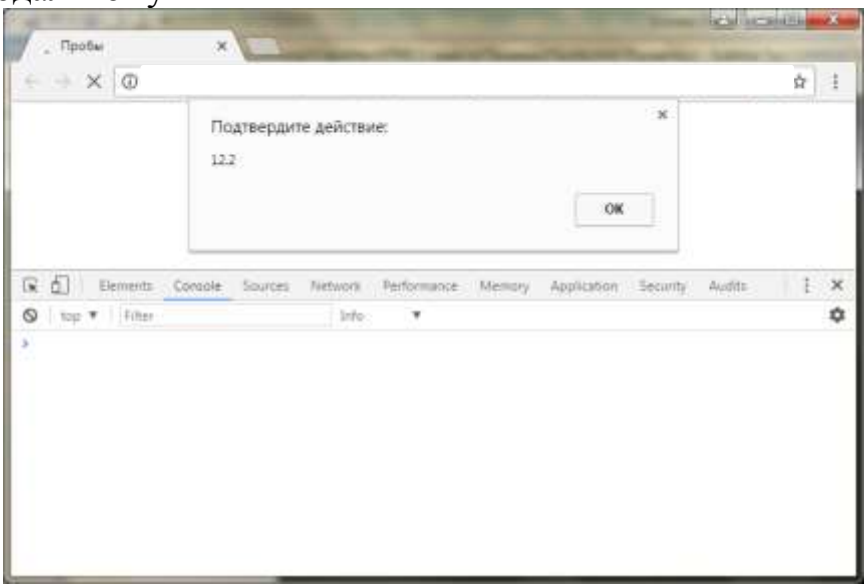
Визначення функції міститься у операторі присвоювання, тобто у виразі.

var name=function(…..) {
…..тіло функції;
}

Наприклад,

```
var count=function (num1,num2,num3){
        var sum=num1+num2+num3;
        return sum;
}
alert(count(5.2,3,4));
```

Результатом буде 12.2 у модальному вікні



В цьому прикладі ми додатково розглянули і застосування оператору **return**, який вказує на змінну, що повертається при звертанні до функції. Оператор return може застосовуватися і сам по собі, у такому разі він вказує, що функція завершена (або здійснено вихід із функції) і управління передається назовні.
Якщо у тілі функції є оператор return, то наступні за ним команди з тіла функції не будуть виконуватися.
При використанні конструкції function expression ОБОВ'ЯЗКОВО звернення до функції має знаходитися ПІСЛЯ визначення функції

Засіб визначення функції у вигляді виразу часто використовують у конструкціях логічних операторів.

Визначення і звертання у одній конструкції

(function(«формальні параметри») {
…..тіло функції;
})(«фактичні параметри»);

При такому способі визначення функція не має імені. «Вся функція» міститься у круглих дужках.

Наприклад,
```
(function (num1,num2,num3){
        var sum=num1+num2+num3;
        console.log(sum);
})(5,12,1);
```
У консолі браузера отримаємо 18.

Рекурсія
У тілі однієї функції можна використовувати (звертатися) до інших функції. Особливим випадком звернення, коли функція у своєму тілі має виклик самої себе, є *рекурсією*.
Рекурсивні функції найбільш складні для використання новачком. Слід уважно вивчити структуру (синтаксис) запису рекурсивної функції і розуміти дії програмного коду при виконанні функції.
Рекурсія використовується у випадках, коли розв'язання одної задачі можна уявити, як певну дію у сукупності з розв'язанням такій же задачі у більш простому варіанті.

Прикладом рекурсії є така математична дія, як зведення у натуральний ступінь $x^n = x \cdot x^{n-1}$

Код JS такої рекурсії:

```
function pow(x, n) {
  if (n != 1) {          // доки n != 1, зводити обчислення pow(x,n) до pow(x,n-1)
    return x * pow(x, n - 1);
  } else {
    return x;
  }
}
// Звернемося до функції для обчислення 2^4
alert( pow(2, 4) );  // отримаємо 16
```

Тут було реалізовано такий алгоритм:

1. `pow(2, 4) = 2 * pow(2, 3)`
2. `pow(2, 3) = 2 * pow(2, 2)`
3. `pow(2, 2) = 2 * pow(2, 1)`
4. `pow(2, 1) = 2`

Функція pow рекурсивно визиває сама себе, поки n не дорівнює 1.
Значення, при якому рекурсія закінчується називають *базисом рекурсії* (в нас базис 1).
Загальна кількість вкладених викликів називають *глибиною рекурсії* (в нашому прикладі звертанні глибина рекурсії 4).
Максимальна глибина рекурсії у браузерах обмежена, для багатьох інтерпретаторів вона становить 10000 (іноді може зустрітися більше).

Хід виконання функції інтерпретатором браузера.

1. Інтерпретатор ініціалізує «пустий об’єкт» функцію. Ініціалізує всі змінні, які визначені у тілі функції. Виділяє для них пам’ять.
2. При визові функції формальним змінним надаються значення, і виконуються команди тіла функції.
3. Після виконання функції об’єкт «викидається», і пам’ять очищується.

При написанні функцій візьміть за правило: одна функція – одна «смислова» дія. Не перевантажуйте функції зайвим.

Область «бачення». Локальні і глобальні змінні

Локальні змінні – це змінні, які визначені у тілі функції за допомогою var, та змінні, які є формальними параметрами функції. Такі змінні доступні самій функції і тим функціям, що в неї вкладені (якщо такі є).
Локальні змінні «живуть» доки виконується функція. Говорять, що вони «є у локальній області бачення».
Блоки умовних і циклічних операторів, які можуть знаходитися у функції, не впливають на область бачення функції.
Існує **правило**: Всі локальні змінні функції, за винятком параметрів функції (формальних змінних) слід визначати на початку функції, а потім використовувати у тілі функції.
Наприклад,

«Поганий» запис	***Запис за правилом***

function cycle(){
  for(var i=0; i<3; i++) {
  var j=i*2;
  }
}
	function cycle(){
	  var i,j;
	  for(i=0;i <3; i++) {
	  j=i*2;
	  }
	}

Глобальні змінні – це змінні, які визначені за межами будь-якої функції.
При виконанні функції інтерпретатор «продивляє» локальну область бачення функції і, якщо не знаходить там «потрібної» змінної, то «підіймається» у глобальну зону бачення, і шукає там.

Наприклад,

```
var userName=’Петро’;
function showMessage(){
  var message=’Мене звати ’+ userName;
  console.log(message);
}
showMessage(); // Результат «Мене звати Петро»
```

У цьому коді локальною є змінна message, а змінна userName – глобальна.

Проте, слід брати до уваги, що **локальні змінні під час виконання функції є пріорітетнішими над глобальними.**

Наприклад,

```
var userName=’Петро’;
function showMessage(){
  var userName=’Тетяна’;
  var message=’Мене звати ’+ userName;
  console.log(message);
}
showMessage(); // результат «Мене звати Тетяна»
```

Примітка. Визначені у глобальному середовищі змінні «живуть вічно» ☺, або їх не перепризначать.
Перепризначити (надати інше значення) глобальну змінну можна і в тілі функції. Для цього слід використати оператор присвоювання без директиви var.

Порівняйте результати у консолі браузера після виконання таких кодів:

var userName=’Петро’;
function showMessage(){
var userName=’Тетяна’;
var message=’Мене звати ’+ userName;
console.log(message);
}
showMessage(); // результат «Мене звати Тетяна»
console.log(userName); // **результат «Петро»**
	var userName=’Петро’;
	function showMessage(){
	userName=’Тетяна’;
	var message=’Мене звати ’+ userName;
	console.log(message);
	}
	showMessage(); // результат «Мене звати Тетяна»
	console.log(userName); // **результат «Тетяна»**

Область бачення у JS коді носить назву **scope.**

Scope – це набір змінних, об’єктів і функції, до яких є доступ із конкретної часті програмного коду.

Глобальна область бачення, її ще називають **window**, визначена для програмного коду в цілому. Змінні, що визначені у window, «живуть» (займають ділянки пам’яті) на протязі виконання всього JS-коду. Вони можуть бути перепризначені у будь-який момент. Але завелика кількість глобальних змінних тільки засмічує пам’ять пристрою, де працює браузер. Тому кількість таких змінних слід зводити до мінімуму.

Локальні змінні визначені у межах функції і «живуть» доки виконується функція. Під час виконання функції визначена локально змінна має перевагу над глобальною.

Кожна вкладена функція має свою scope. А змінні визначені у функції, яка має вкладені функції, доступні всім вкладеним функціям, але НЕ НАВПАКИ!

Імена функції

Обираючи ім’я для функції можна слідувати таким же правилам, як і при наданні імені для змінної, тобто використовувати camelCase і починати з букви, або із символів підкреслення «_» чи долара «$» , але слід пам’ятати, що функція – це дія, тому ім’ям функції рекомендовано обирати дієслово. Так ім’я функції буде забезпечувати більшу читабельність коду в цілому.
Зазвичай, використовуються дієслівні префікси, що позначають загальний характер дії, після яких слідує уточнення:

функції, які починаються з «show» - щось показують, *наприклад*,

```
showMessage(..)        // префікс show - "показати" повідомлення
```

функції, які починаються із «get» - щось отримують, *наприклад*,

```
getAge(..)             //префікс  get - "отримати" вік
```

Наведемо ще прикладів дієслівних префіксів:

```
calcD(..)              // префікс calc, "обчислити" величину D
createForm(..)         // префікс create, "створити" форму
checkPermission(..)    // префікс check, "перевірити" дозвіл
```

Як правило, функція, що має префікс check повертає значення true або false.

І ще раз хочу привернути увагу шановного читача до того, що функція повинна робити тільки те, що явно мається на увазі її назвою. І це повинно бути єдиною метою функції..

Якщо дії для досягнення мети функції доволі складні, то має сенс поділити їх на піддії і кожну піддію виділити у окрему функцію. Найчастіше це робиться для того, щоб краще структурувати код і зробити його більш читабельним.

Але найголовніше - в функції не повинно бути нічого, крім власне дії і піддій, нерозривно пов'язаних з нею. *Наприклад*, функція перевірки даних (скажімо, "validate") не повинна показувати повідомлення про помилку, якщо та зустрінеться. Її дія – перевірити, якщо користувачеві потрібне повідомлення, то воно оформлюється після виклику функції із використанням значень, що нею повернуті.

Об’єктна модель документа (DOM part)

DOM (Document Object Model) – це основний інструмент для роботи і динамічних змін на web-сторінках. Відповідно DOM документ має ієрархічну структуру (структуру дерева). DOM – це надання документа у вигляді дерева об’єктів, які доступні для змін кодами JS.

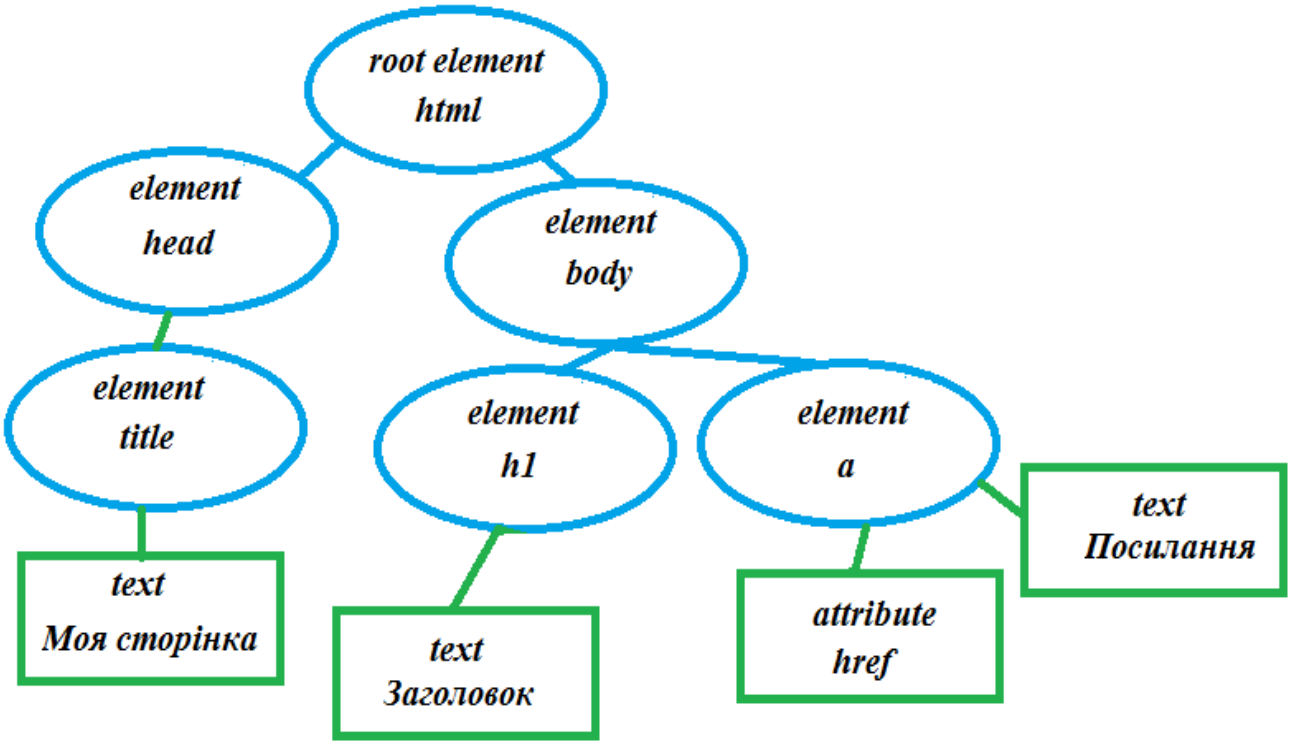
Після завантаження для кожного html- документу браузер складає DOM, тобто об’єктну модель документа цієї сторінки.

Кожен HTML-тег утворює вузол дерева з типом «елемент». Вкладені у цей тег інші теги стають дочірніми вузлами. Для надання тексту утворюється вузол з типом «текст».

Для наступного документа:

```
<html>
  <head>
     <title>Моя сторінка</title>
  </head>
  <body>
     <h1>Заголовок</h1>
      <a href=”#”>Посилання</a>
  </body>
</html>
```

Браузер створить такий DOM

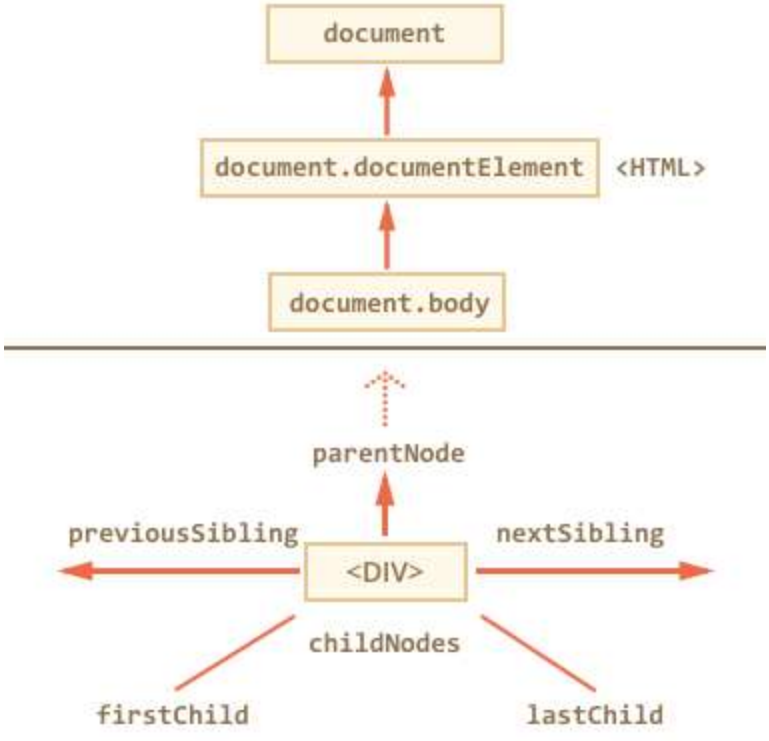


Тут у «овалах» надані вузли (element), а у «прямокутниках» вузли (text).

DOM зберігає взаємозв'язки між елементами, виносить у окремі вузли теги, атрибути тегів, текст, коментарі. Таким чином, до кожного вузла можна надати доступ і змінити його за допомогою JS:

- змінити елемент тег на сторінці;
- змінити атрибути;
- змінити css стилі;
- видалити існуючи на сторінці елементи теги і атрибути;
- додавати елементи теги і атрибути;
- реагувати на події користувача відносно до html- елементів;
- створювати події…..

Для того щоб змінювати DOM елемент до нього слід «доступитися» (звернутися, отримати доступ).
Доступ к DOM починається з об'єкту **document**. З цього об'єкту можна доступитися до будь-яких вузлів.
Доступ (звернення) до вузлів здійснюється за допомогою відповідних методів, які повертають, зазвичай у змінну, посилання на вузол для подальшого використання у JS коді для зміни властивостей і обробці методами. На схемі надано основні посилання, за якими можна переходити по вузлам DOM.



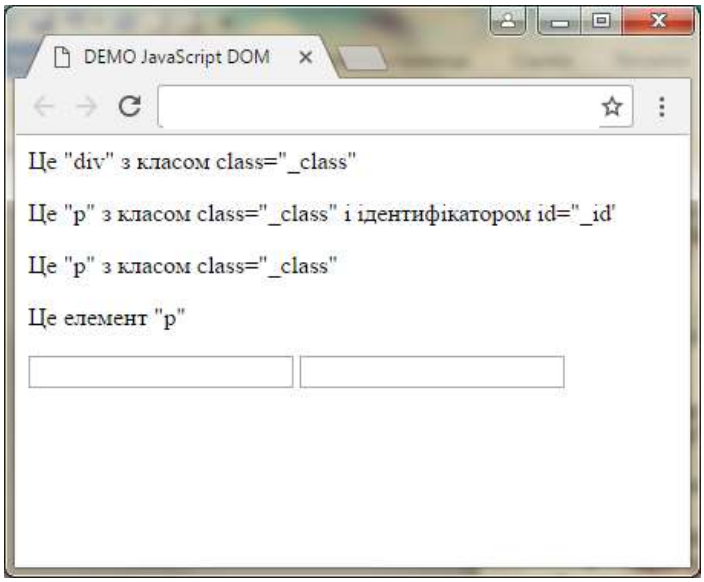
Серед DOM вузлів вирізняють поняття «пращури» і «нащадки», «дочірні елементи», «сусідні елементи». Детально ознайомитися із цим можна на даному ресурсі https://learn.javascript.ru/traversing-dom

У DOM елементів JS є властивості і атрибути. І властивості, і атрибути мають ім'я і значення.

Звернення (доступ) до елементів DOM

Звернення до елементів DOM розглянемо на прикладах.
Створимо html-код, який будемо застосовувати у наших прикладах.

```
<!DOCTYPE html>
<html>
<head>
        <title>DEMO JavaScript
DOM</title>
        <meta charset="utf-8">
</head>
<body>
        <div class="_class">
            Це "div" з класом
class="_class"
        </div>

        <p id="_id" class="_class">
            Це "p" з класом
class="_class" і ідентифікатором
id="_id'
        </p>

        <p class="_class">
           Це "p" з класом
class="_class" </p>

        <p>Це елемент "p"</p>

        <input type="text">
        <input type="password">

        <!-- тут будемо додавати
скріпт -->

</body>
</html>
```



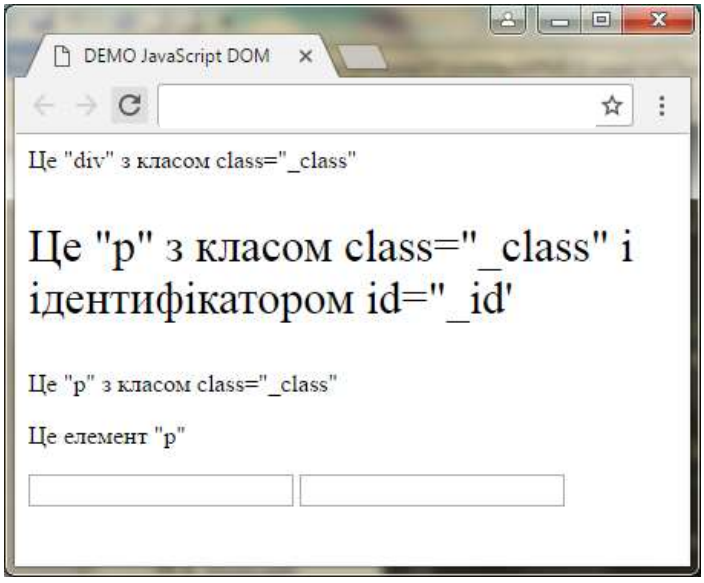
*Приклад 1 (**звернення за ідентифікатором**).*
Змінити розмір шрифта у елемента із ідентифікатором «_id».

Ми будемо використовувати метод **.getElementById** – це метод об'єкта **document**. Він повертає посилання на вузол документа, яке можна використовувати для зміни властивостей і звертатися до методів цього вузла. Виділимо у змінну idPerem елемент з ідентифікатором за допомогою вказаного методу. А потім використаємо цю змінну, надав їй стиль розмір шрифту, як значення відповідної властивості.

Такий скріпт буде мати вигляд:

```
<script type="text/javascript">
	var idPerem=document.getElementById('_id');
	idPerem.style.fontSize="30px";
</script>
```

І у браузері побачимо:

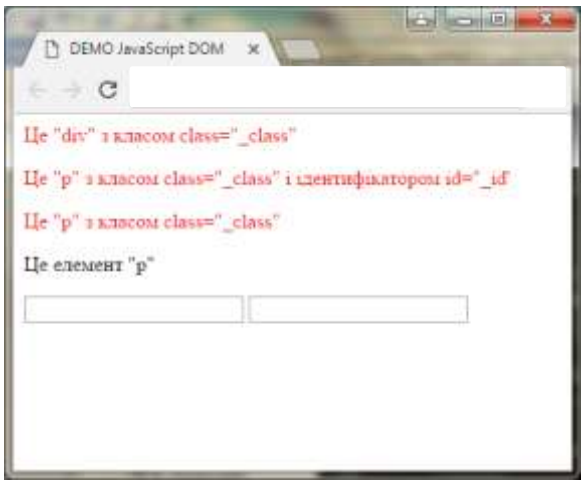


Зверніть увагу!
Синтаксис властивостей стилів CSS має деякі розбіжності. Властивість, яка у CSS пишеться через дефіс, у JS ми записуємо за правилом camelCase, *наприклад:*

CSS	JavaScript
font-size	fontSize

*Приклад 2 (**звернення за класом**).*
Всім елементам з класом «_class» (в нашому прикладі їх три) змінимо колір на красний.

```
<script type="text/javascript">
	var classArr=document.getElementsByClassName("_class");
	for (var i = 0; i < classArr.length; i++) {
		classArr[i].style.color="red";
}
</script>
```



І у браузері побачимо

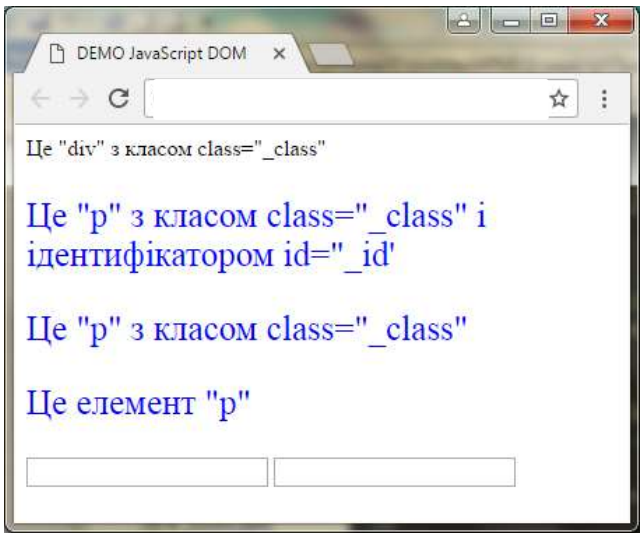
У цьому прикладі ми зверталися до кількох елементів із класом «_class», тому записали getElement**s**, і **з**мінна «classArr» є масивом із трьох елементів. Тому для зміни кольору кожному елементу використовується цикл for.

*Приклад 3 (**звернення за тегом**).*

Елементам параграф (тег «p») задамо колір синій і змінимо розмір шрифту. Параграфів у нашому прикладі три. Тому за аналогією із прикладом 2 будемо оперувати із масивом.

Створимо скріпт:

```
<script type="text/javascript">
        var tagArr=document.getElementsByTagName("p");
        for (var i = 0; i < tagArr.length; i++) {
                tagArr[i].style.color="blue";
                tagArr[i].style.fontSize="25px";
}
</script>
```



У браузері

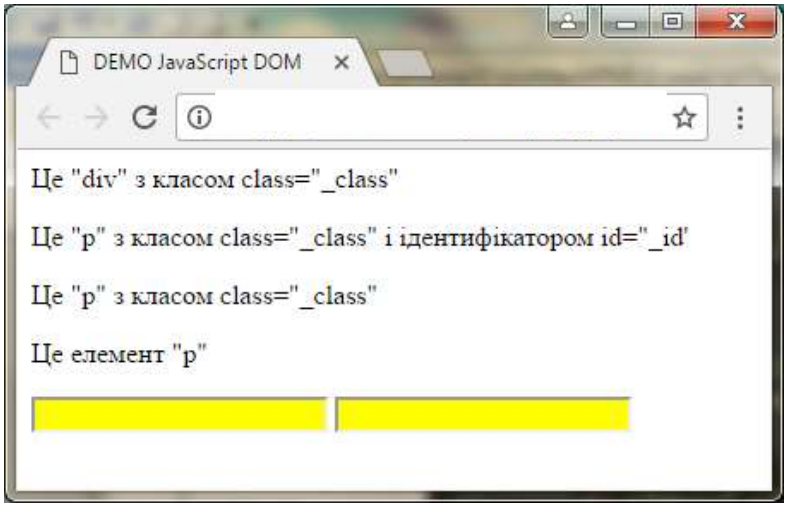
За необхідністю знайти на сторінці всі елементи, які відповідають певному CSS селектору (id , тип , клас , атрибут, значення атрибуту) можна використати метод **.querySelectorAll**

*Приклад 4 (**всі селектори** input)*
Змінимо колір фону елементів input на жовтий.

Запишемо скріпт:

```
<script type="text/javascript">
	var InputElements=document.querySelectorAll('input');
	for (var i = 0; i < InputElements.length; i++) {
		InputElements[i].style.backgroundColor="yellow";
}
</script>
```

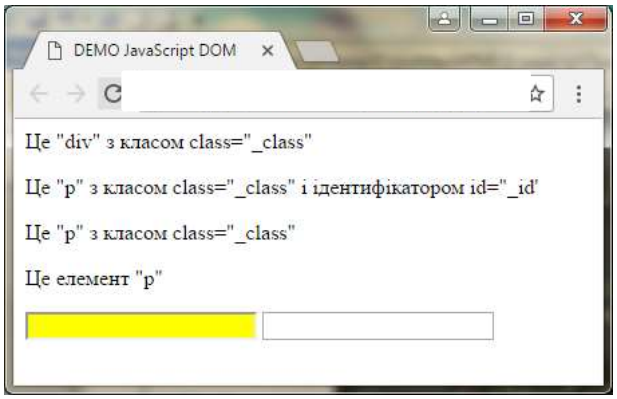
У браузері



Використання методу **.querySelector** надає можливість оперувати з першим з вказаного селектора, який зустрівся на сторінці.

Порівняйте результат прикладу 4 із результатом такого скріпту:

```
<script type="text/javascript">
	var elements=document.querySelector('input');
	elements.style.backgroundColor="yellow";
</script>
```



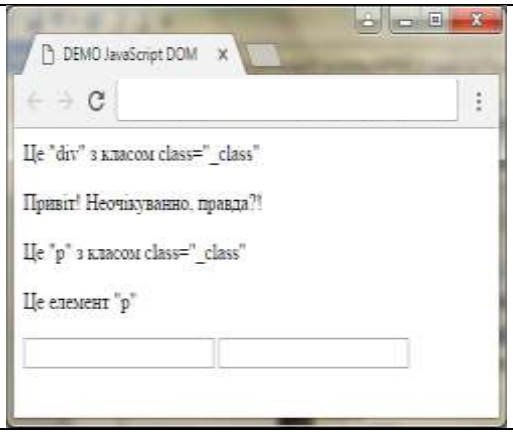
Увага!
Не можна отримати доступ до елементу, якого ще не існує в момент виконання скріпту.

Встановлення нового значення для текстового вузла. Властивість innerHTML

Приклад 5

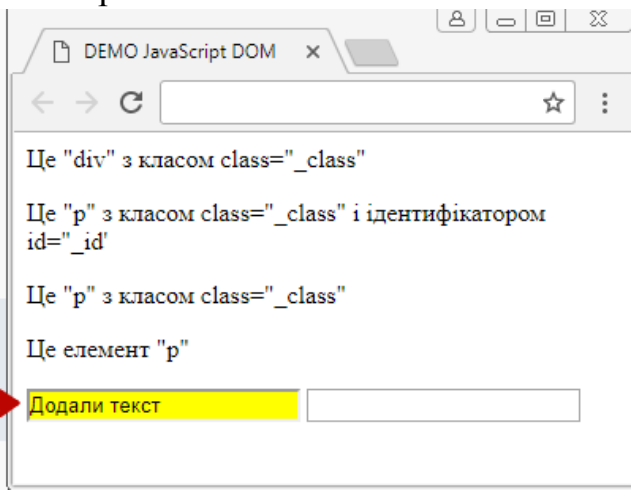
Доступимося до елемента з ідентифікатором «_id'» і змінимо текстовий вузол, тобто контент параграфа:

```
<script type="text/javascript">
var peremId=document.getElementById('_id');
peremId.innerHTML="Привіт! Неочікуванно, правда?!";
</script>
```



Але текстовими є і значення деяких атрибутів. Приміром , для того, щоб додати текст у текстове поле елемента input, треба скористатися властивістю **value**:

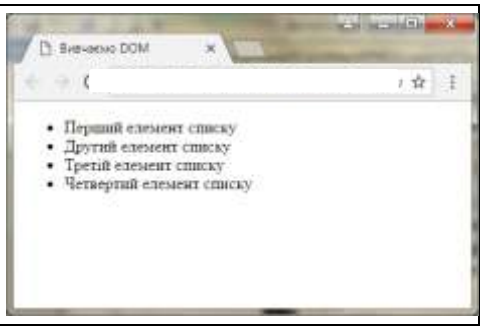
```
<p class="_class">
    Це "p" з класом class="_class" </p>

<p>Це елемент "p"</p>

<input type="text">
<input type="password">

<!--  тут будемо додавати скріпт -->
<script type="text/javascript">
    var element=document.querySelector('input');
    element.style.backgroundColor="yellow";
    element.value='Додали текст';

</script>
```



Додавання, видалення і заміна DOM елементів

Приклади дій додавання, видалення і заміни розглянемо на елементі маркованого списку. Для цього створимо html-сторінку (наведемо тільки вміст елементу body стандартного html-коду):

```
<ul id="list">
        <li>Перший елемент списку</li>
        <li>Другий елемент списку</li>
        <li>Третій елемент списку</li>
        <li>Четвертий елемент списку</li>
</ul>
<!-- тут будемо додавати скріпт -->
```



Візьмемо до уваги таку ієрархічну підпорядкованість вузлів нашої сторінки:
«li» - дочірній елемент до елементу «ul»
«ul» - пращур (батьківський) елементу «li»

Приклад 6 (**додавання елементу наприкінці** списку **appendChild**)

Додавання нового елементу – це процедура з кількох шагів:

1. Створимо елемент «li»:
```
var child=document.createElement("li");
```

2. Створимо текстовий вузол:
```
var text=document.createTextNode("Пятый элемент");
```

3. Приєднаємо (дочірнім) текстовий вузол до нового елементу:
```
child.appendChild(text);
```

4. Отримаємо доступ до батьківського елементу (до списку, який є на сторінці) за його id:
```
var parent=document.getElementById("list");
```

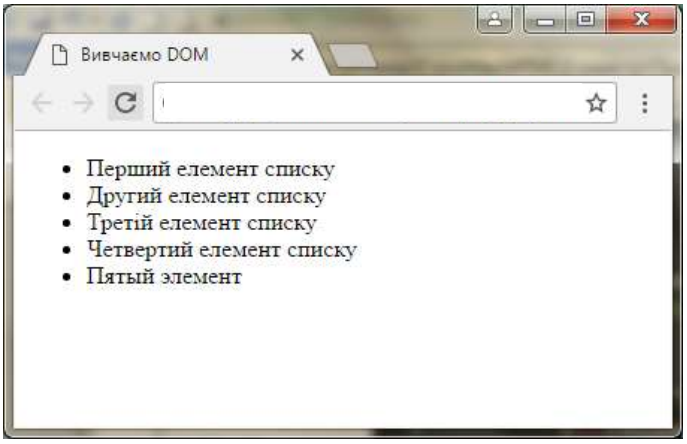
5. Приєднаємо «новий li» до «свого батька» (parent):
```
parent.appendChild(child);
```

Властивість **appendChild** додає елемент у кінець «елементу батька». У нашому прикладі наприкінці списку.

Остаточно JS-код має вигляд:

```
var child=document.createElement("li");
var text=document.createTextNode("Пятый элемент");
child.appendChild(text);
var parent=document.getElementById("list");
parent.appendChild(child);
```

А змінена ним сторінка:

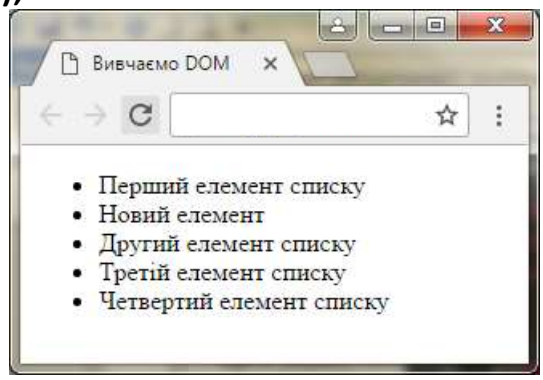


Приклад 7 (**додавання елементу «у середину»** списку **insertBefore**)
Властивість **insertBefore** додає елемент ПЕРЕД вказаним елементом. Але ж для цього нам потрібно доступитися до всіх елементів «li», організував із них масив. Тоді можна буде «вказати» потрібний «li» за індексом.

Скріпт, який додасть «Новий елемент» перед елементом масиву за індексом 1:

```
var child=document.createElement("li");
var text=document.createTextNode("Новий елемент");
child.appendChild(text);
var parent=document.getElementById("list");
var liItems=document.getElementsByTagName("li");
parent.insertBefore(child, liItems[1]);
```

створить таку web-сторінку

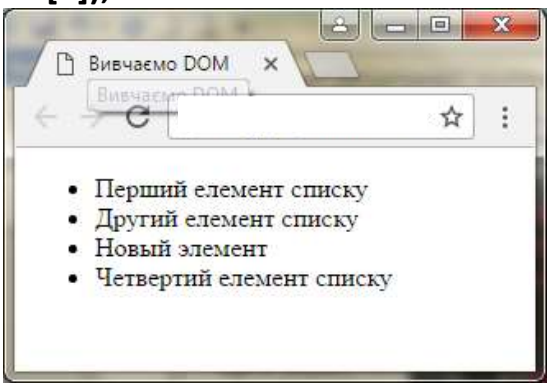


Приклад 9 (**заміна елементів** списку **replaceChild)**
Замінимо третій елемент списку на створений елемент списку з текстом «Новий елемент»

Відповідний скріпт буде таким:

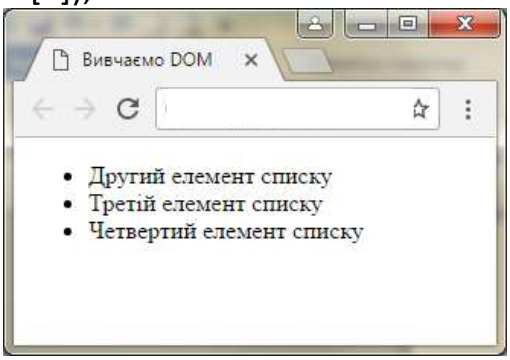
```
var child=document.createElement("li");
var text=document.createTextNode("Новый элемент");
child.appendChild(text);
var parent=document.getElementById("list");
var liItems =document.getElementsByTagName("li");
parent.replaceChild(child, liItems[2]);
```

А web-сторінка стане такою



Приклад 9 (**видалення елементів** списку **removeChild)**
Видалимо перший елемент списку. Для цього застосуємо **removeChild** до елементу масиву liItems із індексом 0:

```
var parent=document.getElementById("list");
var liItems=document.getElementsByTagName("li");
parent.removeChild(liItems[0]);
```



Створення нового вікна браузера і контенту засобами JS

Засобами JS можна створювати нові браузерні вікна. Вікно «попап» (від англ. popup) – один із найстаріших засобів показати користувачу новий документ, створюється методом **window.open**.
Синтаксис

var newWin = window.open(url, name, params);

аргументи метода:
url - URL для завантаження у нове вікно. Якщо контент вікна буде створюватися теж засобами JS, то на цьому місці може бути "about:blank", або навіть пуста строка ""
name – ім'я нового вікна, наприклад, "hello", або пуста строка ""
params – конфігурація нового вікна. Параметри конфігурації записуються через кому «,» **без пробілів**!

left/top - задається числом, координати верхнього лівого кута вікна відносно екрану. Не може позиціюватися за межами екрану;

width/height - задається числом, ширина і висота вікна

наприклад, вікно розміром 100×100.

```
var newWin = window.open("about:blank","","width=100,height=100");
```

Якщо у методі не вказати жодного аргумента, то буде створено не вікно, а вкладка, *наприклад*,

```
var newWin = window.open();
var newWin = window.open(https://learn.javascript.ru/window-methods);
```

або так `var win=window.open('Перша проба.html');`

якщо файл Перша проба.html розташований на локальному комп'ютері. Треба вказувати повну адресу для файлу.

Можна задавати і інші аргументи, які описують властивості вікна (наявність меню, полос прокрутки, панель навігації браузера тощо). Дефолтні значення цих аргументів «no».

Розглянемо *приклад* створення нового вікна із контентом у вигляді відформатованого параграфу. Скріпт надамо у вигляді функції:

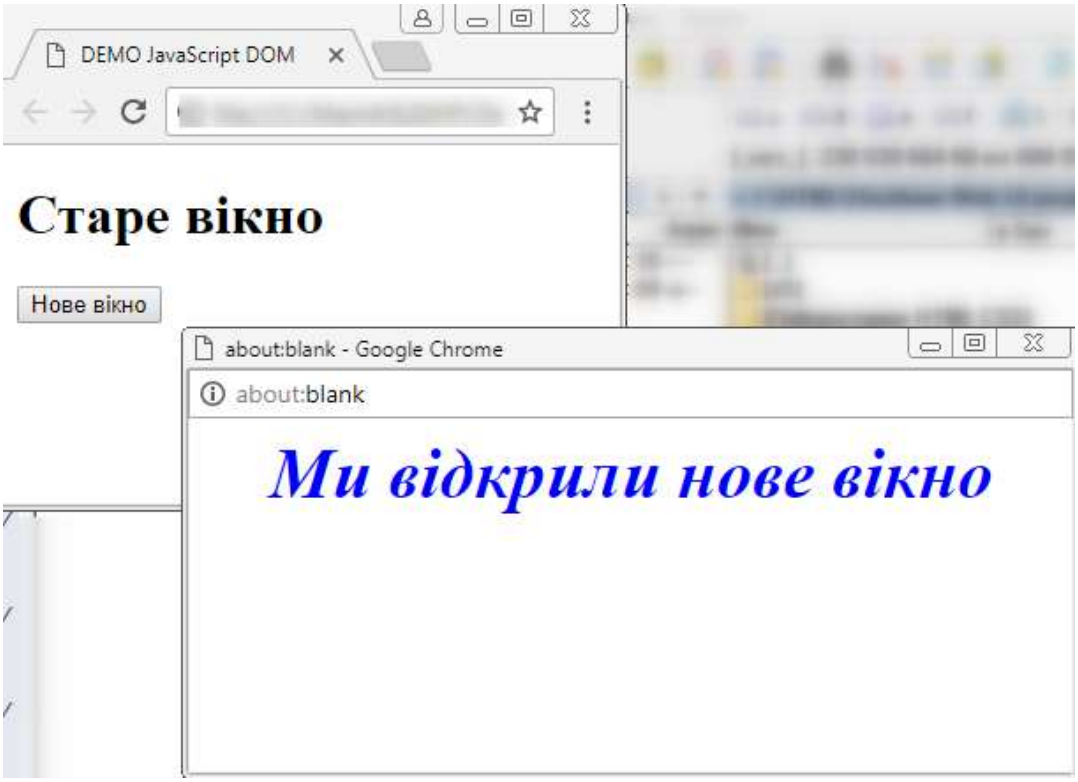
```
function showNewWindow() {
// відкриємо нове вікно
var newWin=window.open('about:blank','myWin','top=200,left=150,width=500,height=200');

// створюємо елемент параграф
        var newWinPargraph = newWin.document.createElement('p');

// додаємо текст у створений елемент
        newWinPargraph.innerHTML="Ми відкрили нове вікно";

// додамо стилів для елементу параграф
        newWinPargraph.style.color = 'blue';
        newWinPargraph.style.textAlign='center';
        newWinPargraph.style.fontSize='40px';
        newWinPargraph.style.fontWeight = 'bold';
        newWinPargraph.style.fontStyle = 'italic';

// "вставимо" параграф у нове вікно дочірнім елементом у body
        newWin.document.body.appendChild(newWinPargraph);
}
```

Для того, щоб перевірити роботу функції, створіть стандартний html-файл. Додайте, *наприклад*, заголовок «Старе вікно» і кнопку. З кнопкою зв'яжемо подію onClick (детальніше про події дивись у наступному розділі), тобто клік мишею по кнопці буде запускати нашу функцію:

Елемент <body> html-коду:

```
<body>
        <h1>Старе вікно</h1>
        <form>
        <input type="button" value="Нове вікно" onClick="showNewWindow();">
        </form>
</body>
```

Клікнув мишею по кнопці Нове вікно ми побачимо:



Додавати стилі елементам нового вікна можна і створив новий елемент style, записав його дочірнім у елемент head нового вікна. Тоді замість операторів «додамо стилів…»

```
// додамо стилів для елементу параграф
    newWinPargraph.style.color = 'blue';
    newWinPargraph.style.textAlign='center';
    newWinPargraph.style.fontSize='40px';
    newWinPargraph.style.fontWeight = 'bold';
    newWinPargraph.style.fontStyle = 'italic';
```

можна написати такий код:

```
// Створюємо елемент і додаємо стилів
    var newWinStyle=newWin.document.createElement('style');
        newWinStyle.innerHTML='p { color: blue; text-align: center; font-size: 40px;
            font-weight: bold; font-style: italic }';

//  "вставимо" стиль у нове вікно дочірнім елементом у head
    newWin.document.head.appendChild(newWinStyle);
```

Зауваження. *Що використовувати* createTextNode або innerHTML
Результат виконання може бути різним:
innerHTML додасть саме HTML тегі, якщо вони є у ‘тексті’
createTextNode додає виключно текстовий вузол, все інтерпретує, як текст

Події

Для реакції на дії користувача і внутрішню взаємодію скріптов існують *події*.
Подія – це сигнал браузера про те, що щось відбувається, і його можна застосовувати у функціях JS.
Події зазвичай використовуються спільно із функціями, тоді функція не буде виконуватися доти не відбудеться відповідна їй подія.

Приклади дій користувача, що інтерпретуються, як події:

- кліки мишею (по кнопці або тексту на сторінці);
- переміщення миші над певною зоною Web-сторінки;
- натискання клавіш на клавіатурі;
- завантаження Web-сторінки, зображення чи відео;
- відправлення html-форми;

тощо

Наведемо JS перелік найбільш «популярних» подій:

Події «миші»:

- ✓ **click** – відбувається, коли клікнули на елементі лівою кнопкою миші
- ✓ **contextmenu** – відбувається, коли клікнули на елементі правою кнопкою миші
- ✓ **mouseover** – відбувається, коли миша наводиться на елемент
- ✓ **mousedown** або **mouseup** – коли кнопку миші натиснули або відпустили (віджали)

Натискання або «віджимання» клавіші на клавіатурі – це події **keydown** або **keyup**, відповідно.

- ✓ **mousemove** – при проведенні миші (рух мишею)

Події на елементах керування:

- ✓ **submit** – користувач відправив форму <form>
- ✓ **focus** – користувач «сфокусувався» на елементі, наприклад клікнув поле `<input>`

Існує і багато інших подій.

Для того, щоб отримати обробник, події , до її ім'я слід додавати «on».

Існує декілька способів призначати обробник конкретної події елемента.
Розглянемо найбільш популярні:

- **Обробник події записати прямо утегі**, що відкривається.
  Наприклад, реакція на клік мишею по елементу кнопка запишеться так:

```
<input type='button' value='Клікни мене' onclick=alert('Дяка!');>
```

Тут JS-код пишеться у лапках в одну строку.
Такий спосіб написання обробника дуже простий і зручний, тому часто використовується у невеличких простих задачах. Причому події можна присвоїти і функцію, яка описана у коді сторінки, *наприклад* так

```
<input type='button' value='Перевірка' onclick='chAll();'>
```

Тут мається на увазі, що функція chAll() існує у скріпті сторінки.

- **Обробник події надається у вигляді функції**. Для знайомства із цим способом наведемо приклад.

Розглянемо події «наведення мишею» і «клік мишею»:
(пояснення дивись у коді)

```
<!DOCTYPE html>
<html>
<head>
	<title>Вивчаємо події в JS</title>
	<meta charset="utf-8">
</head>
<body>
	<h1 id="title">Заголовок</h1>
	<p id="text">Клікни цей текст</p>
	<script type="text/javascript">

// реагує на проведення мишею над Заголовком
		document.getElementById("title").onmousemove=function(){
			this.style.color="yellow";
	};

// реагує на клік миші по тексту параграфу
		document.getElementById("text").onclick=function(){
			this.innerHTML="Oooops!";
			this.style.color="red";
	};
	</script>
</body>
</html>
```

Створіть html-файл з цим кодом і подивіться, як він працює.

Використання бібліотеки jQuery

Точно так, як CSS виокремлює візуалізацію web-сторінки від структури HTML, jQuery виокремлює поведінку (динаміку) web-сторінки від структури HTML. Бібліотека jQuery містить функціональність, яка корисна для широкого кола задач.

Основна ідея jQuery полягає у тому, щоб зробити код JavaScript коротшим і простішим. Іноді, одна строка коду з використанням jQuery може замінити 20 строк коду JS.

Для використання бібліотеку jQuery слід підключити до HTML-коду як «звичайний» файл скріпту.

```
<script type="text/javascript" src="jQwery/jquery-3.1.1.min.js"></script>
```

Слід брати до уваги, якщо до HTML-коду додається кілька файлів скріптов, то бібліотеку jQuery треба підключити першою.

Бібліотека постійно оновлюється, свіжу версію можна завантажити із сайту http://jquery.com/

Якщо web-сторінка виконується на комп'ютері із постійним підключенням до мережі Інтернет, то бібліотеку jQuery можна підключити за допомогою CDN.
CDN (Content delivery network), «мережа доставки контенту» - це множина серверів із спеціалізованим програмним забезпеченням, які пришвидшують доставку («віддачу») контенту кінцевому користувачеві. При завантаженні сторінки за допомогою CDN буде знайдено найближчий сервер, на якому є jQuery, і саме звідки і буде підключено бібліотеку.

Підключення скріпту за допомогою CDN, наприклад:

```
<script type="text/javascript" src="//code.jquery.com/jquery-1.11.3.min.js"></script>
```

Функція jQuery

Основною функцією в бібліотеці є функція **jQuery()** , за допомогою якої виконуються основні дії на web-сторінці : пошук, маніпулювання елементами.
Ця **функція повертає об'єкт jQuery.**
Для зручності **замість jQuery() записують $()**

Наприклад,
$('div') – функція, що збирає зі сторінки всі блоки div
(аналог методу document.getElemetsByTagName('div') з «чистого» JS коду).
Тому, зазвичай, приблизний синтаксис оператора jQuery має вигляд:

$('селектор').дія('властивість дії');

селектор – елемент чи елементи, з якими виконується дія;
дія – що саме буде відбуватися із обраними елементами. У якості дії ми можемо додати певний ефект, css-стиль, змінити html-код тощо. Також тут можуть бути вказані події.
властивість дії – вказують, якщо вони передбачені дією.

Зверніть увагу на символ «$», саме він вказує на те, що використовуються функції бібліотеки jQuery.

Доступ до елементів DOM у jQuery

Елемент	«Чистий» JavaScript	jQwery
з ідентифікатором, *наприклад* id=’title’	document.getElementById('title')	$('#title')
з класом, *наприклад* class=’text’	document.getElementByClass('text')	$(‘.text’)
з тегом, *наприклад* <p>	document.getElementsByTagName(‘p’)	$("p")
Обрати всі елементи		$(‘*’)
Обрати вкладену конструкцію, *наприклад*, всі посилання <a>у блоці		$(‘div a’)
Обрати всі параграфи <p>, блоки і заголовки <h1>		$('p, div, h1')
Обрати прямих нащадків, *наприклад*, всі параграфи <p>, які є у елементах з класом one		$('.one>p')

Наприклад, створимо html-код, підключив бібліотеку jQuery

```
<!DOCTYPE html>
<html>
<head>
        <title>Знайомимся із jQwery</title>
        <meta charset="utf-8">
```

Треба обрати одне з підключень

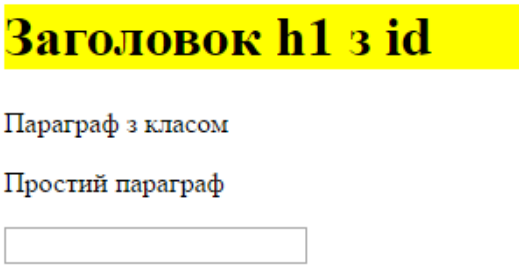
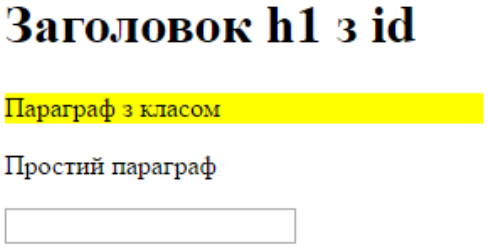
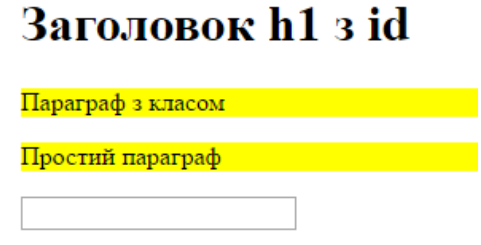
```
<!--Підключення бібліотеки через CDN -->
<script type="text/javascript" src="//code.jquery.com/jquery-1.11.3.min.js"></script>
```

```
<!-- Підключення локальної бібліотеки -->
        <script type="text/javascript" src="JS/jquery-3.1.1.min.js"></script>
```

```
        <script type="text/javascript">
```

```
        </script>
</head>
<body>
  <h1 id="title">Заголовок h1 з id</h1>
  <p class="text">Параграф з класом</p>
  <p>Простий параграф</p>
  <input type="text">

<!— Підключення файлу власного скріпта-->
<script type="text/javascript" src="JS/MyScript.js"></script>

</body>
```

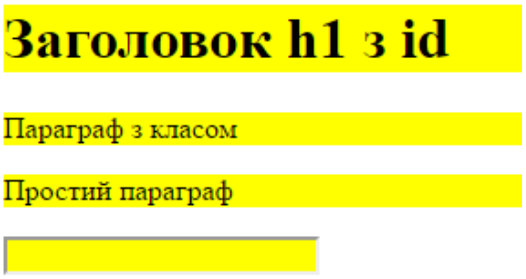
У файлі MyScript.js папці JS запишіть такі оператори і подивиться, як вони змінюють елементи сторінки із ідентифікатором і класом, відповідно

Скріпт	***Відображення у браузері***
$("#title").css("background-color","yellow");	
$(".text").css("background-color","yellow");	
$("p").css("background-color","yellow"); **Зверніть увагу**, нам не прийшлося використовувати цикл для того, щоб змінити колір кільком параграфам. Ми це зробили одним оператором із функцією jQuery	

Можна за допомогою функції jQuery без використання циклу змінити колір фона у різних елементів на сторінці (звернувшись за ідентифікатором, тегом і елементом input)
такий оператор запишемо так:

```
$("#title,p,input[type='text']").css("background-color","yellow");
```

У браузері побачимо:



Розглянемо приклади застосування функції jQuery до вкладених елементів.

Для цього завернемо у тег слово «параграф», яке є у контенті тегов <p>:

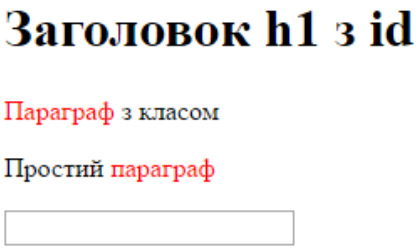
```
<p class="text"><span>Параграф</span> з класом</p>
  <p>Простий <span>параграф</span></p>
```

Змінимо колір «цього слова» на красний. Це можна зробити двома засобами:

Засіб 1 `$('p span').css("color", "red");`

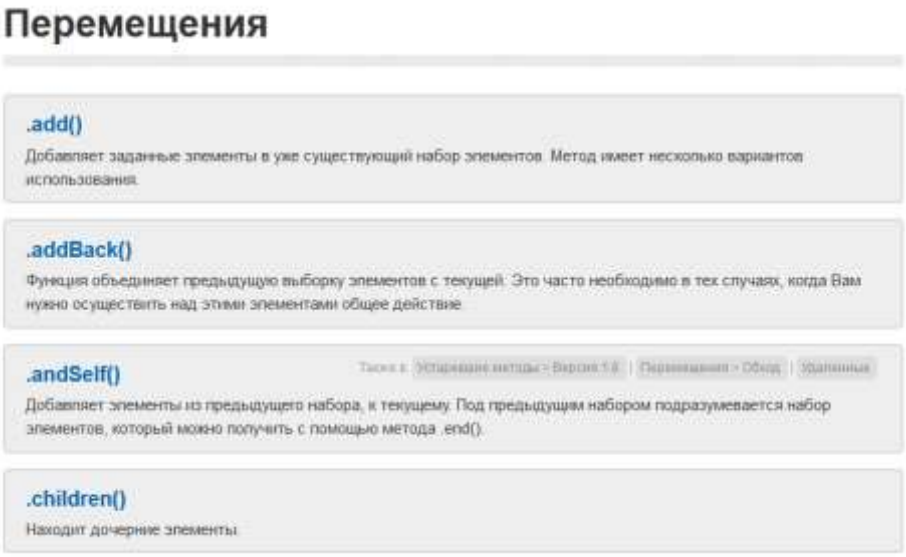
Засіб 2 `$('p').find('span').css("color", "red");`

У браузері це виглядатиме так



У документації jQuery можна знайти і інші засоби доступу до вкладених елементів, а також дочірніх, батьківських, подібних, сусідніх.

Наведемо фрагмент розділу Перемещения з сайту https://jquery-docs.ru/category/traversing/



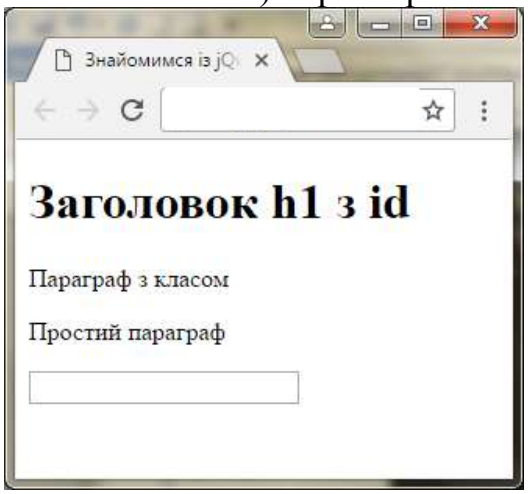
Функції $() і події

Розглянемо на прикладах події «клік мишею» по елементах сторінки із попереднього підрозділу

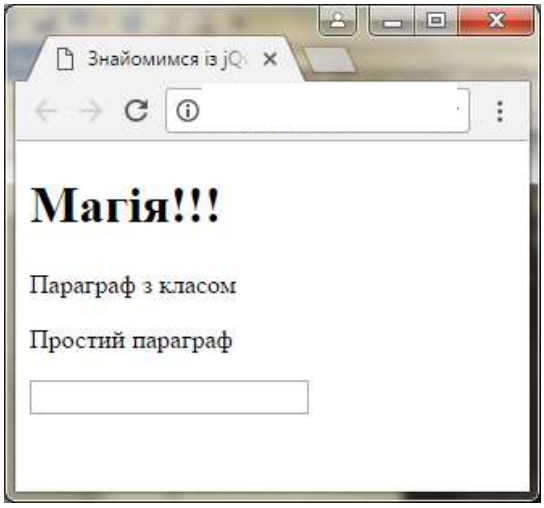
Приклад 1 (**клік по текстовому елементу**)
Якщо написати такий скріпт:

```
$("#title").click(function() {
  $("#title").text("Магія!!!");
});
```

То клік мишею по елементу з ідентифікатором id=”title” (на нашій сторінці це заголовок <h1 >) перетворить сторінку



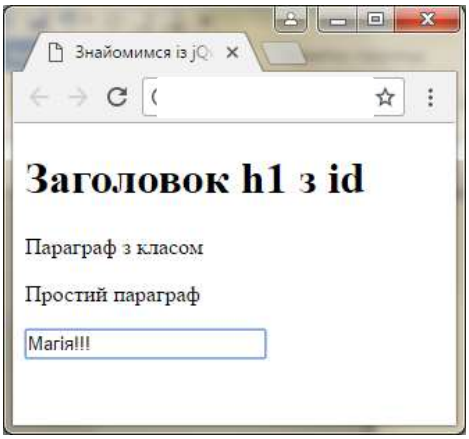
у таку



Приклад 2 (**клік по полю елемента input**)
Наступний скріпт

```
$('input').focus(function() {
  $('input').val("Магія!!!");
});
```

при клікі мишею по полю «стандартну сторінку перетворить на таку



Більше про функції подій можна дізнатися із розділу События довідкової документації jQuery

«Ланцюжок методів» у jQuery. Chaining

«Розгорнутий вигляд»	«Ланцюжок методів»

$('#title').**bind**('click',function() {
  $('#title').text('Магия!!!');
  $('#title').css('color','red');
  $('#title').css('font-size','20px');
});
	$('#title').bind('click',function() {
	  $('#title').text('Магия!!!').css('color','red').css('font-size','20px');
	});
	або у такому вигляді:
	$('#title').bind('click',function() {
	  $('#title').text('Магия!!!')
	  .css('color','red')
	  .css('font-size','20px');
	});

bind – встановлює обробник подій для всіх елементів набору jQuery.

Для набуття навичок написання кодів з функціями jQuery рекомендується спочатку розробити код на «чистому» JS, а потім переписати його «у jQuery».

Лабораторні роботи і самостійні завдання

Для набуття навичок по створенню власних доданків пропонуємо виконати наступні завдання.

Перед виконанням завдань дайте відповіді на питання:

1. Де може бути розташований скріпт у html-коді сторінки, як він виконується?
2. Який синтаксис підключення скриптів до сторінки, якщо:
   а) скріпт є частиною html-коду;
   б) скріпт міститься у окремому файлі *.js ?
3. Який синтаксис «базових» операторів мови JavaScript?
4. Що таке DOM part і як отримати доступ до елементів сторінки?
5. Як комунікувати із користувачем за допомогою модальних вікон?
6. Де можна подивитися на можливі помилки при роботі коду? Які засоби «дебажінгу» вам відомі?

Завдання 1

Знайти кількість подільників для числа, що вводить користувач, і обчислити їх суму.

Пояснення до коду:
При написанні коду будемо використовувати застосування користувачем модальних вікон.
Операція «%» - надає залишок від ділення числа, аналогічна операції «mod» мови Visual Basic.

Самостійно:
Створіть html-сторінку із базовою конфігурацією у кодуванні UTF-8.
Запишіть код скріпту і протестуйте його дію. Примірний вигляд вмісту html-файлу Завдання 1 поданий нижче.

Спробуйте самостійно створити скріпт, який буде давати відповідь «Так» чи «Ні» на запитання: «Чи однакова кількість подільників у чисел A і B, які ввів користувач?»

Примірний алгоритм нового завдання:

1. отримати від користувача два числа;
2. обчислити кількість подільників кожного з введених чисел;
3. порівняти кількість подільників;
4. сформувати і надати користувачу відповідь.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Завдання 1</title>
    <meta charset="utf-8">
</head>
<body>
<h1><small>Завдання 1</small></h1>
<p>Знайти кількість подільників числа і їх суму</p>
<script type="text/javascript">
    var number=+prompt('Введіть число','');
    var count=0;
    var summa=0;
    for (var i=1; i<=number;i++) {
        if (number % i == 0) {
            summa=summa+i;
            count++;
        }
    }
    alert('Кількість подільників = '+count);
    alert('Сума подільників = '+summa);
</script>

</body>
</html>
```

Завдання 2

Створення форми для перевірки знань таблиці множення ☺
Якщо у наданій формі відмічені всі вірні приклади, то у новому вікні буде написано «Вірно», у противному випадку побачимо «Не вірно»

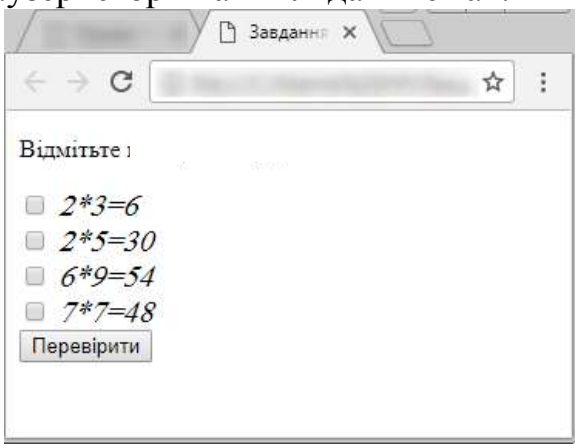
Пояснення до коду:
Створимо web-сторінку із обраними прикладами для перевірки знань:

```html
<body>
    <p> Відмітьте всі вірні твердження:</p>
    <form>
        <label><input type="checkbox" id="cb1"> 2*3=6</label>
        <label><input type="checkbox" id="cb2"> 2*5=30</label>
        <label><input type="checkbox" id="cb3"> 6*9=54</label>
        <label><input type="checkbox" id="cb4"> 7*7=48</label>
        <input type="button" value="Перевірити" onClick="checkAll();">
    </form>
</body>
```

Для того, щоб елементи label починалися із нового рядка, додамо стиль:

```
    label {
    display: block;
    font-style: oblique;
    font-size: 20px;
}
```

Звертання до функції checkAll() зв’язано із подією click об’єкту button
При відображені у браузері сторінка виглядатиме так:



У елемент head сторінки додамо скріпт – функцію checkAll():

```
<script type="text/javascript">
    function checkAll() {
    var ans='Не вірно';
// доступимося до елементів DOM по id
    var p1=document.getElementById('cb1');
    var p2=document.getElementById('cb2');
    var p3=document.getElementById('cb3');
    var p4=document.getElementById('cb4');
// перевірка встановлених і не встановлених флажків
    if (p1.checked && !p2.checked && p3.checked && !p4.checked) ans='Вірно';
// вивод результата в нове вікно
// змінна newWin екземпляр об'єкта window
// створюється  пусте вікно розміром 120 на 120,
    var newWin=window.open("about:blank","","width=120,height=120");
// створюються елементи нового вікна body і заголовок h2
    var newWinBody = newWin.document.body;
    var myMessage=newWin.document.createElement('h2');
// встановлюється нове значення текстового вузла у заголовку
    myMessage.innerHTML=ans+"!!!";
// елемент заголовок myMessage додається у елемент body newWinBody
    newWinBody.appendChild(myMessage);
    }
</script>
```

Складіть і протестуйте код.
Нове вікно може виглядати так , якщо будуть обрані вірні відповіді

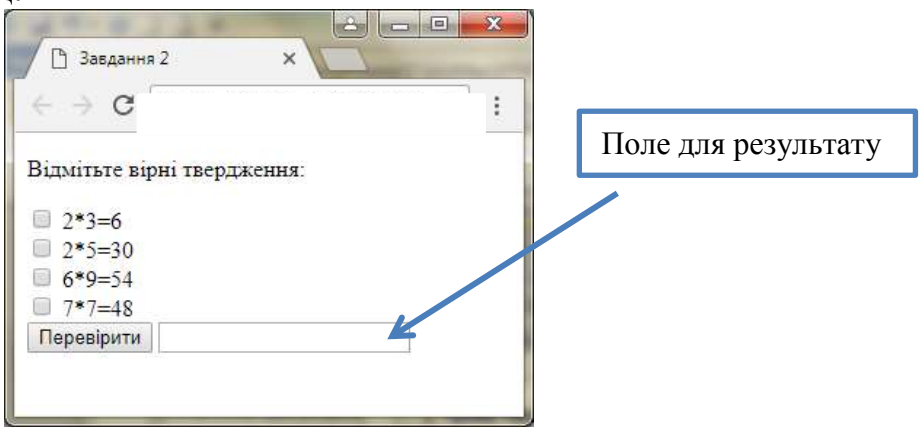
Самостійно:

Створить динамічну сторінку, коли відповідь «Вірно/Не вірно» з'являється не у новому вікні, а у елементі «текстове поле». Такий доданок можна зробити із попереднього коду, відредагував його наступнім чином:

- До html-коду сторінки слід додати елемент «текстове поле» поряд із кнопкою «Перевірити», а саме записати

```
<input type="text"id="result" value="">
```

Сторінка матиме вигляд:



- У JS коді функції checkAll() слід прибрати все, що відноситься до створення і запису результату у нове вікно.

І на цьому місці записати:

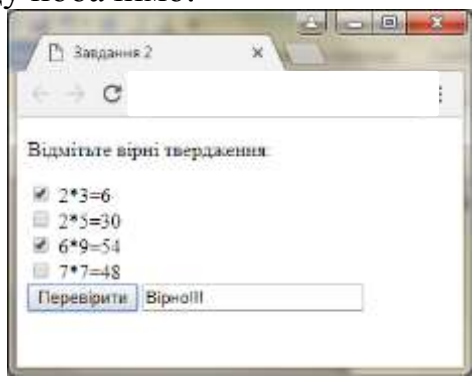
- Доступимося до елементу «текстове поле», наприклад, так

```
var fieldTxt=document.getElementById('result');
```

- Тоді вивід результату у поле запишемо у вигляді:

```
fieldTxt.value=ans+'!!!';
```

Після відпрацювання коду побачимо:



Спробуйте розробити функцію, яка «очищує» перелік перед новим вибором відповіді, і додайте кнопку, із подією Click якої зв'язана функція.

Завдання 3

Створимо додаток, який дає можливість обрати із переліку «особу» і надати її «фото» у вікні браузеру. У якості «фото» будуть використані збережені у папці Gif відповідні файли.

Пояснення до коду:
Спочатку створимо html- сторінку з елементом «перелік» і «кнопкою». У

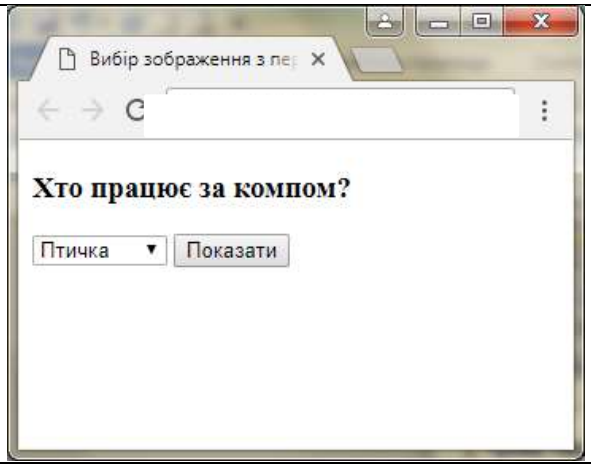
відкритому тегі елемента button додамо обробник події onClick.

До елементів переліку додамо ідентифікатори.

HTML код

```
<!DOCTYPE html>
<html>
    <head>
        <title>Вибір зображення з переліку</title>
        <meta charset="utf-8">
        <script src="JS/JS-3.js"></script>
    </head>
    <body>
        <h3>Хто працює за компом?</h3>
        <form>
            <select>
                <option id='Bird'>Птичка</option>
                <option id='Manky'>Мавпочка</option>
                <option id='Huck'>Хакер</option>
            </select>
            <input type="button" value="Показати" onclick="showFoto();">
        </form>
    </body>
</html>
```

У браузері



Створимо функцію showFoto() у файлі JS-3.js , який розміщений у папці JS

File Edit Selection Find View Goto Tools Project Preferences Help

3 - New(Список и картинки).html × JS-3.js ×

```
function showFoto() {
        // створення нових елементів
    var newPargraph = document.createElement('h2');
    var newImage = document.createElement('img');
        // атрибути картинки
    newImage.alt = 'Тут буде картинка';
    newImage.width = 200;
        // доступ до елементу за відповідним Id
    var selectBird=document.getElementById('Bird');
    var selectManky=document.getElementById('Manky');
    var selectHuck=document.getElementById('Huck');
        // визначення обраного елементу у списку
    if (selectBird.selected) {
        newPargraph.innerHTML=selectBird.text ;
        newImage.src ='C:/My Project/Gif/Bird.gif';
    }
    else if (selectManky.selected) {
        newPargraph.innerHTML=selectManky.text;
        newImage.src ='C:/My Project/Gif/Manky.gif';
    }
    else if (selectHuck.selected) {
        newPargraph.innerHTML=selectHuck.text;
        newImage.src ='C:/My Project/Gif/Progman.gif';
    }
        // додавання нових елементів у існуюче вікно
    document.body.appendChild(newPargraph);
    document.body.appendChild(newImage);
}

```

Line 1, Column 1 Tab Size: 4 JavaScript

Зверніть увагу, що посилання на файл із картинкою треба надавати у вигляді «повного імені файлу».

Запишіть і протестуйте доданок.

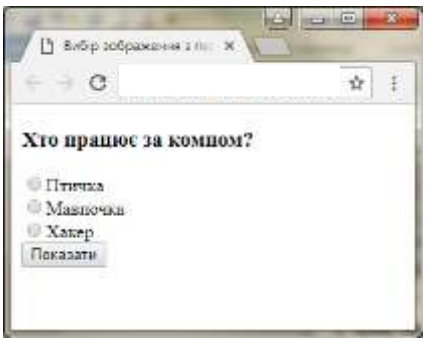
Самостійно:

Створіть доданок за мотивами розглянутого прикладу, у якому картинка обирається не за допомогою переліку, а за допомогою «радіокнопки», елемент

<input type=’radio’>

А метод, за яким визначаємо активну радіо кнопку

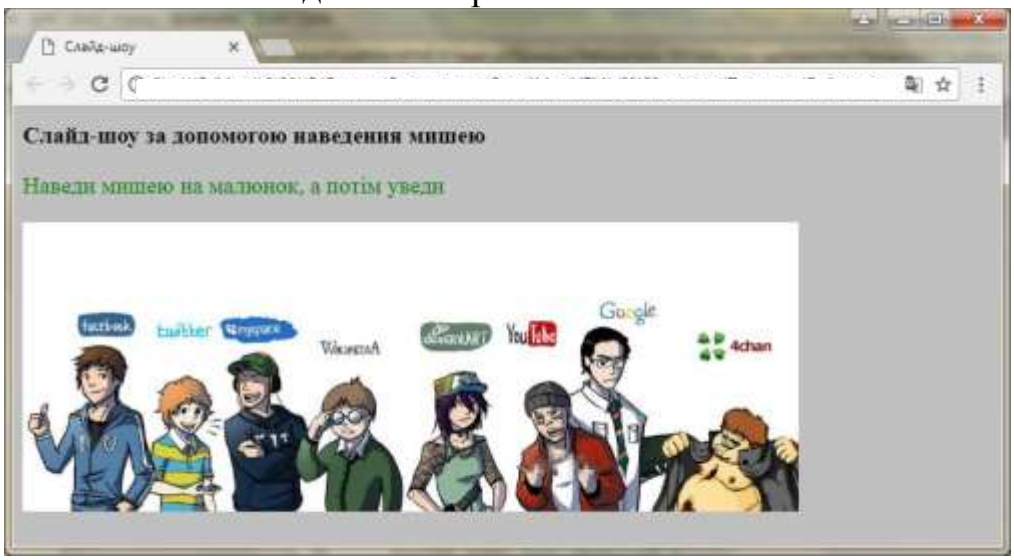
. checked



Завдання 4

Створимо доданок, у якому картинка, що є на web-сторінці, заміняється на іншу картинку, якщо наводити не неї мишею. І повертає «стару» картинку, якщо уводити мишу від картинки. Файли jpg, що використовуються у коді, зберігаються у папці Jpg.

Пояснення до коду:
Напишемо html-код такої сторінки



Html-код такої сторінки має вигляд:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Слайд-шоу</title>
        <meta charset="utf-8">
        <style type="text/css">
            body {background-color: silver;}
            h1,p {font-size: 20px;}
            p {color: green;}
            #Picture {width: 80%;}
        </style>
        <script type="text/javascript">

            // тут буде скрипт function doEvent(type)

        </script>
    </head>
    <body>
        <h1>Слайд-шоу за допомогою наведення мишею</h1>
        <p>Наведи мишею на малюнок, а потім уведи</p>
        <!-- у відкритому тегі додани обробники подій -->
        <img id="Picture" src="Jpg/Соцсети люди.jpg" onMouseOver="doEvent(true);"
        onMouseOut="doEvent(false);" >
        </body>
</html>
```

Зверніть увагу, що у елемент head доданий елемент style, який містить CSS-код для стилю web-сторінки. А у відкритому тегі img містяться обробники подій onMouseOver і onMouseOut

Напишемо скріпт функції doEvent(type)

```
function doEvent(type) {
// доступимося до об'єкту DOM за id 'Picture'
var cv=document.getElementById('Picture');

/* якщо параметр функції дорівнює true, то
     змінюємо властивість src объекта cv на Эйнштейн вечеря.jpg
     У протилежному випадковіі показуємо "стару" картинку */
cv.src=type==true ? "Jpg/Эйнштейн вечеря.jpg" : "Jpg/Соцсети люди.jpg";
}
```

У скріпті використано логічний оператор if у скороченій формі запису.

Коли користувач буде наводити мишею на картинку, то web-сторінка набуватиме такого вигляду:

Запишіть і протестуйте доданок.

Самостійно:
Відредагуйте створений скріпт таким чином, щоб при наведенні мишею на картинку змінювалася не тільки картинка, а і фон елементу body. При уведенні миші «із картинки» web-сторінка набувала «старого» вигляду.
Пам'ятайте, для того щоб змінювати фон елементу body до нього треба доступитися, наприклад за тегом:

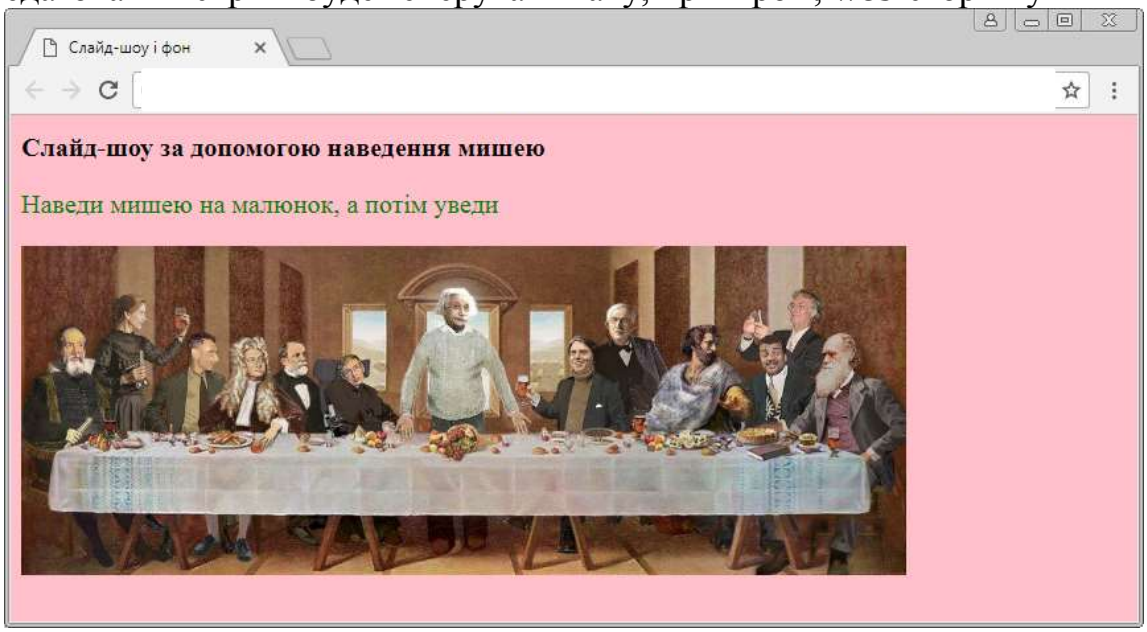
```
var bd=document.getElementsByTagName('body');
```

У такому разі змінна bd є змінною масиву, тому використовувати її треба із зазначенням індексу, наприклад:

```
bd[0].style.backgroundColor="pink";
```

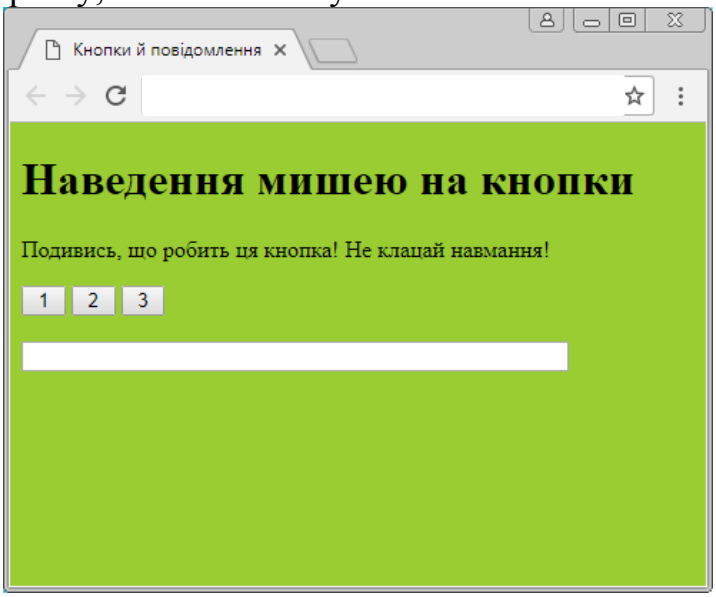
А логічний оператор if слід записати у стандартному вигляді, тому що за умовою (true / false) треба виконувати більш, ніж один оператор, і скорочена форма оператора if для цього не підходе.

Відредагований скріпт буде генерувати таку, приміром, web-сторінку



Завдання 5

Створимо web-сторінку, як на малюнку

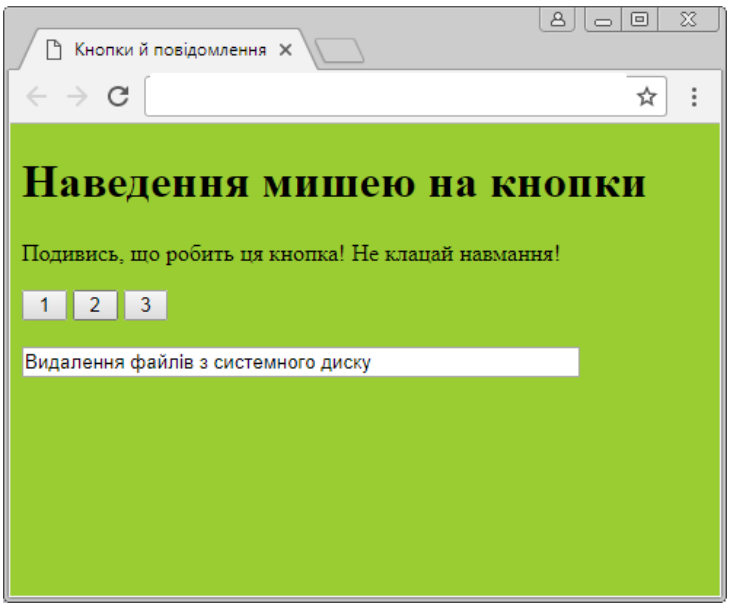


Розробимо доданок, який при наведенні мишкою на кнопку у текстовому полі «показує» надпис-«підказку», що робить ця кнопка

Пояснення до коду:
Масив mess зберігає всі можливі написи-«підказки», а його елементом із індексом 0 є «пуста строка», яка з'являється, коли миша відводиться від кнопки.
Формальний параметр num функції doEvent(num) при звертанні до функції замінюється на номер кнопки.

```
<head>
    <title>Кнопки й повідомлення</title>
    <meta charset="utf-8">
    <style>
        body{background-color: yellowgreen;}
    </style>
    <script>
        function doEvent(num){
        // визначимо масив "повідомлень" і його значення
        var mess = ["","Завершення роботи комп'ютера","Видалення файлів з системного диску", "
            Форматування вінчестеру"];
        // доступимося до об'єкту текстове поле з id info
        var cv=document.getElementById('info');
        // змінюємо властивість value об'єкту cv на відповідний елемент масиву
        cv.value= mess[num];
        }
    </script>
</head>
<body>
    <h1>Наведення мишею на кнопки</h1>
    <p>Подивись, що робить ця кнопка! Не клацай навмання!</p>
    <form>
        <input type="button" value=" 1 " onMouseOver="doEvent(1);" onMouseOut="doEvent(0);">
        <input type="button" value=" 2 " onMouseOver="doEvent(2);" onMouseOut="doEvent(0);">
        <input type="button" value=" 3 " onMouseOver="doEvent(3);" onMouseOut="doEvent(0);">
        <br><br>
        <input type="text" id='info' size="50">
    </form>
</body>
```

Створіть і від тестуйте наданий код. *Наприклад*, при наведенні на кнопку 2 wed-сторінка приймає вигляд:



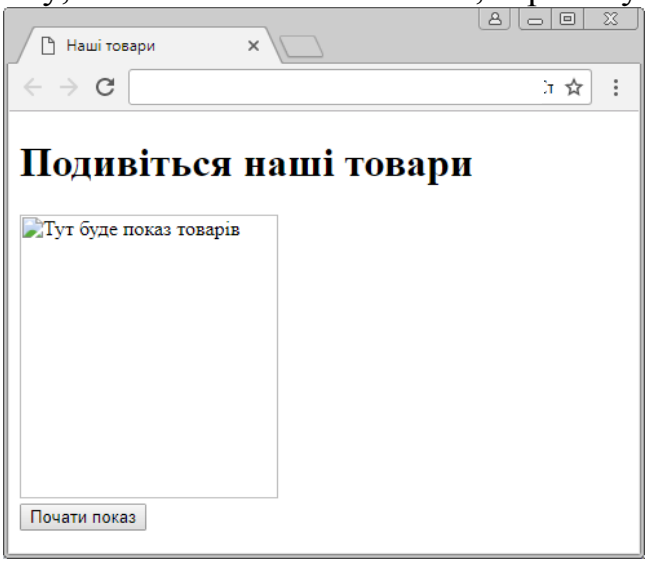
Самостійно:
Розробить доданок, який у текстове поле виводить «підказку» текстом красного кольору у вигляді фрази «Ви навели курсор миші на кнопку …..» (і буде вказано номер кнопки),
Доданок можна отримати, відредагував код Завдання 5:

масив mess не потрібен , і його треба прибрати з коду;
змінити властивість змінної cv можна за допомогою формування фрази із постійною і змінною частинами, наприклад:

cv.value=’ Ви навели курсор миші на кнопку ’ + num;

Завдання 6

Створимо web-сторінку, яка містить заголовок h1, картинку і кнопку



Коди елементів head і body цієї сторінки будуть мати вигляд:

```
<head>
    <title>Наші товари</title>
    <meta charset="utf-8">
    <style type="text/css">
        #slide {width: 200px; height: 220px;}
    </style>
    <script type="text/javascript">
        // тут буде функція slideShow()
    </script>
</head>
<body>
    <h1>Подивіться наші товари</h1>
    <img src="#" id="slide" alt="Тут буде показ товарів">
    <form>
        <input type="button" value="Почати показ" onClick="slideShow();">
    </form>
</body>
```

Створимо функцію slideShow() , яка після кліка по кнопці у «віконці картинки» показує 4 картинки . Картинки при показі змінюються автоматично через декілька секунд.
У нашій функції буде використано метод **setTimeout (func , delay)** забезпечує затримку виконання коду -func на delay мілісекунд («delay» 1000 дорівнює 1 секунді)

```
<script type="text/javascript">
    var  i=0;
    function slideShow() {
    // масив картинок товарів
    var goods = ['Png/Рюкзак.PNG','Png/Китель BDU.PNG','Png/Свитер форменный.PNG','Png/
        Жилет охотника.PNG'];
    // Перевіримо, чи не вийшов індекс за останній елемент масиву
    if (i >3)   i = 0; // якщо вийшов, то лічильник обнуляємо
    //доступимося до об'єкту по id slide
    var showImg=document.getElementById('slide')
    // Меняем свойство src этого объекта на элемент из массива
    showImg.src=goods[i];
    i++;   // збільшуємо лічильник на 1
    // звертаємося до тієї ж функції кожні 1500 мс
    var timer=setTimeout("slideShow()",1500);
    }
</script>
```

Створіть і протестуйте код.

Зверніть увагу, що «наше шоу» нескінченне. Картинки будуть автоматично змінюватися по колу, доти ви не відновите сторінку або не зачините вікно браузера.
Для того, щоб відмінити виконання методу setTimeout використовують той факт, що метод у «своїй змінній» генерує числовий id, який можна використати у методі **clearTimeout().**

Самостійно:
Відредагуйте код таким чином, щоб клік по кнопці «Припинити показ» припиняв показ товарів.
Для цього на web-сторінку треба додати ще одну кнопку, *наприклад*

```
<input type="button" value="Завершити показ" onClick="showOff();">
```

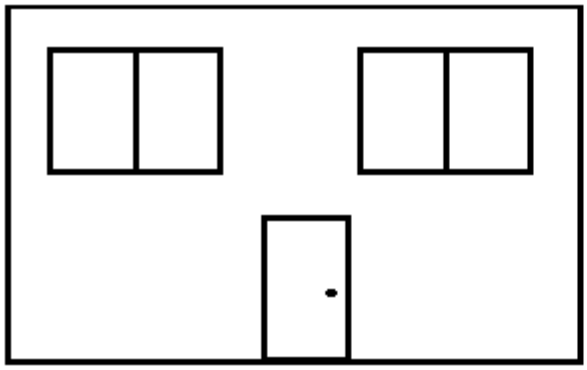
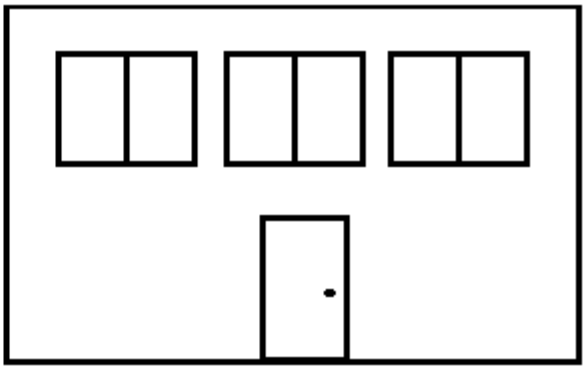
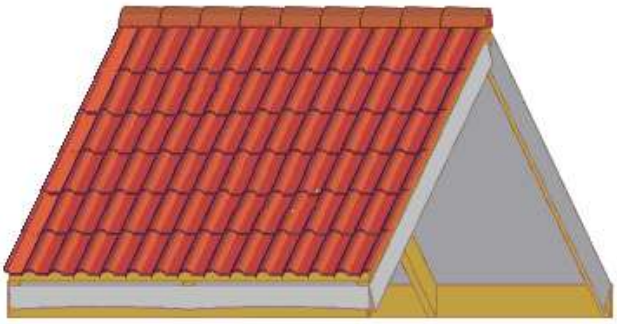
І оформити функцію showOff() , у якій буде задіяний метод **clearTimeout()**

```
clearTimeout(timer);
```

Але **слід пам'ятати** про «зону видимості» змінної timer і відредагувати скріпт таким чином, щоб вона була доступна обом функціям (slideShow() і showOff()).
Зауваження. Функція showOff() може містити всього один оператор, тоді показ товарів буде припинений, а «картинка» товару залишиться на сторінці.

Завдання 7

В графічному редакторі ми створили малюнки-заготовки майбутнього дома. Два малюнка для «фасаду» і два малюнка для «даху». Чотири файли малюнків png, збережені у папці Png

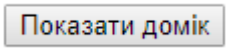





Розробимо проект, який дозволяє комбінувати «фасад» із «дахом» і виводити сконструйований домік у нову вкладку браузеру.

Код елементу <body> нашого проекту буде мати вигляд:

```
<body>
    <form>
        <h3>Оберіть дах</h3>
        <select>
            <option id='roof1'>Черепиця</option>
            <option id='roof2'>Очерет</option>
        </select>
        <br>
        <h3>Оберіть кількість вікон</h3>
        <select>
            <option id='window1'>2 вікна</option>
            <option id='window2'>3 вікна</option>
        </select>
        <br><br>
        <input type="button" value="Показати домік" onclick="Constract()">
    </form>
</body>
```

У атрибуті src елементу <script>, якій розмістимо у елементі <head>, дамо посилання на скріпт функції Constract(), яка визивається кліком по кнопці



:

```
function Constract() {
        // Отримаємо доступ до елементів "даху"
            var elRoof1=document.getElementById('roof1');
            var elRoof2=document.getElementById('roof2');
        // Отримаємо доступ до елементів "вікна"
            var elWin1=document.getElementById('window1');
            var elWin2=document.getElementById('window2');
        // Створення нових елементів
            var roof=document.createElement('img');
            var fasad=document.createElement('img');
            var delimiter=document.createElement('br');
        // атрибути нових елементів
            roof.width=200;
            roof.alt='Картинка даху';
            fasad.width=200;
            fasad.alt='Картинка фасаду';
        // Визначення які конструкції обрано
            if (elRoof1.selected) {
                roof.src='C:/My Project/Png/Cher.png'; }
            if (elRoof2.selected) {
                roof.src='C:/My Project/Png/Kam.png'; }
            if (elWin1.selected) {
                fasad.src='C:/My Project/Png/Win2.png'; }
            if (elWin2.selected) {
                fasad.src='C:/My Project/Png/Win3.png'; }
        // Відкриття нової вкладки у браузері
            var newWin=window.open();
        // Надання сформованих елементів у нове вікно браузера
            newWin.document.body.appendChild(roof);
            newWin.document.body.appendChild(delimiter);
            newWin.document.body.appendChild(fasad);
    }
```

Елементи img є лінійними, тому ми додали перехід на нову строку (елемент br), щоб «фасад» розташовувався з нової строки.

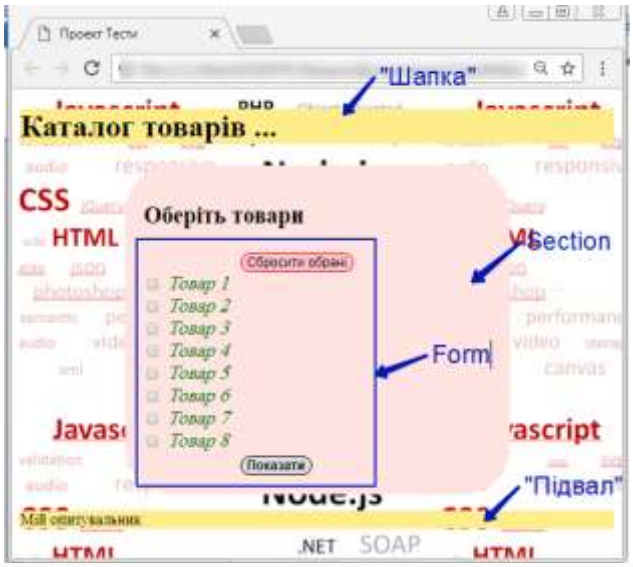
Створіть і протестуйте проект.

Самостійно:
Додайте до проекту можливість обирати вигляд доміка з трьох варіантів даху і фасаду. По-перше, розробіть у графічному редакторові картинки «дахів» і «фасадів». Слідкуйте, щоб картинки були однакового розміру за шириною. Можете відтворити сконструйований домік у тому ж вікні браузеру, або замість нової вкладки створити нове вікно надав його параметри.

Завдання 8

Розробимо код, який надає користувачу інформацію про виділені на сторінці «флажки».

Розробимо **код сторінки**:




```
<!DOCTYPE html>
<html>
    <head>
        <title>Проект Тести</title>
        <meta charset="utf-8">
        <link rel="stylesheet" type="text/css" href="CSS/For 8.css">
    </head>
    <body>
        <header>
            <h1>Каталог товарів ...</h1>
        </header>
        <section>
            <h2>Оберіть товари</h2>
            <form class="selection">
                <input type="reset" name="reset" value="Сбросити обрані">

                <label><input type="checkbox"> Товар 1</label>
                <label><input type="checkbox"> Товар 2</label>
                <label><input type="checkbox"> Товар 3</label>
                <label><input type="checkbox"> Товар 4</label>
                <label><input type="checkbox"> Товар 5</label>
                <label><input type="checkbox"> Товар 6</label>
                <label><input type="checkbox"> Товар 7</label>
                <label><input type="checkbox"> Товар 8</label>

                <input type="button" value="Показати" onclick="showRezult();">
            </form>
        </section>
        <footer>
        <p>Мій опитувальник</p>
        </footer>

        <script type="text/javascript" src="JS/JS-8.js"></script>

    </body>
</html>
```

Стилі

Функція, зв'язана з подією Click

Скріпт

Додамо стилів:

```
body {
        background-image: url("../Png/Фон.png");
    }
    header, footer {
        background-color: #ffec8b;
    }
    section {
        width: 400px;
        margin: 0px auto 0px auto;
        padding: 20px;
        box-sizing: border-box;
        border-radius: 50px;
        background-color: mistyrose;
    }
    .selection {
        font-size: 20px;
        font-style: italic;
        color: green;
    }
    label {
            display: block;
            font-style: oblique;
    }
    input[type="reset"] {
        position: relative;
        left: 100px;
        border-radius: 10px;
        border: 1px solid red;
        box-sizing: border-box;
        background-color: pink;
    }
    input[type="button"] {
        position: relative;
        left: 100px;
        border-radius: 10px;
        border: 1px solid black;
        box-sizing: border-box;
        font-weight: bold;
    }
```

До правил двох останніх селекторів, які описують кнопки, можна додати правило: outline: none; тоді не буде з'являтися «блакитна обводка» навколо кнопки при отриманні нею фокуса.

Напишемо скріпт:

Функція showRezult() містить у собі виклик і тіло функції checkElements(). Функція checkElements() збирає зі сторінки всі елементі input за допомогою DOM part методу .querySelecrorAll. Слід мати на увазі, що першим і останнім елементом input у формі є елементи-кнопки:

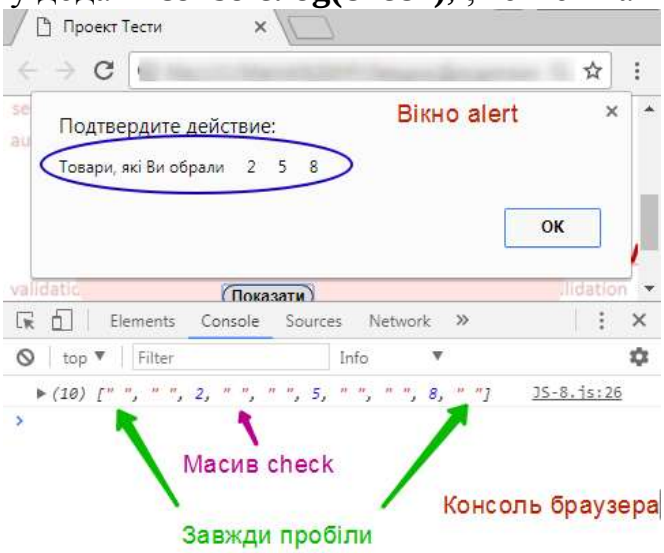
type=reset для очищення обраних елементів checkbox;

type=button для виконання скріпту,

які обробляти не потрібно.

В результаті обробки формується змішаний масив check, значеннями якого є «номера» обраних товарів, або символи пробілу. Індекс масиву співпадає з

«номером» товару. Наприклад, якщо були обрані Товар 2 , Товар 5 і Товар 8, то в результаті обробки масив check складається з таких значень:
Для користувача формується вікно alert у якому надається текстова змінна, що має постійну «Товари, які Ви обрали» и змінну result частини.
Якщо у текст скріпту додати **console.log(check)**; , то можна побачити у консолі:

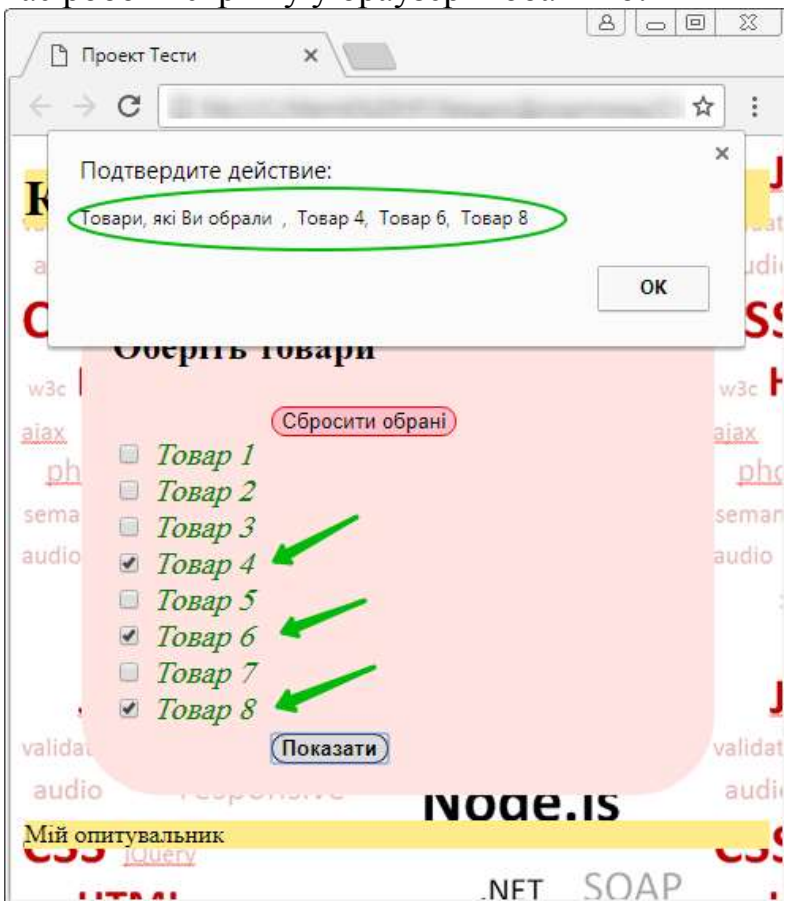


Остаточний вигляд коду скріпта:

```
function showRezult() {
// Звернення до функції, яка перевіряє відповіді
    var rezult;
    checkElements();
// Генерування відповіді у модальне вікно
    alert('Товари, які Ви обрали ' + rezult);

    function checkElements() {
    // Визначити обрані "флажки"
    var i;
// Масив для збереження обраних "флажків"
    var check = [];
    check[0] = ' ';

/* Елементи input сторінки зберемо у масив
Перший і останній елементи цього масиву обробляти не потрібно,
тому що це кнопки, а ні "флажки" */
    var inputArr = document.querySelectorAll('input');
    var k = inputArr.length;
    check[k-1]=' ';

    for (i = 1; i < k; i++) {
//  Сформуємо масив з номерів обраних "флажків", решта пробіли
        check[i] = inputArr[i].checked ? i : ' ';
    }

// Перетворення масиву у строку із розділителем пробіл
    rezult = check.join(' ');

// Повернемо відповідь
        return rezult;
    }
}
```

Створіть проект та протестуйте його роботу.

Самостійно:
У попередньому проекті створювався масив, значення якого потім перетворювалися на строку символів. Але можна й відразу формувати змінну, оминаючи створення та перетворення масиву. Такий проект пропонується створити Вам, шановний читач.
Формалізуємо завдання. Нехай в нас є така сторінка, яка розглядалася у Завданні 8. Створіть скріпт, який у модальне вікно alert надає інформацію про назви товарів, які були обрані користувачем сторінки.
Наприклад, під час роботи скріпту у браузері побачимо:



Якщо буде час та натхнення ☺ , можна попрацювати над дизайном сторінки, написав свій файл CSS стилів.

Рекомендації для JS-коду:

1. за допомогою DOM part слід доступитися до елементів input і label. Це можна зробити методом .querySelectorAll().
2. Створити змінну масиву і записати в нього назви товарів. Тут буде цикл for,

```
    var lblArr = document.querySelectorAll('label');
    var lblLen = lblArr.length;
// Заповнення даними масиву lblTxt
    for (i = 0; i < lblLen; i++) {
        lblTxt[i] = lblArr[i].textContent;
    }
```

наприклад

3. Створити змінну, у якій будемо формувати результат, «накопичуючи» обрані товари за допомогою конкатенації (склеювання). Можна надати їй первинне значення «пробіл», *наприклад* `var goods = ' ';`
4. У циклі for перебрати усі «флажки», визначив, які з них були обрані (checked). І для обраних «флажків» додати назву товару у змінну результату, *наприклад*

```
// Визначемо, які "флажкі" обрані і "додамо" назву товару
    for (i = 1; i < inputLen; i++) {
        if (inputArr[i].checked) {
            goods = goods + ', ' + lblTxt[i-1];
        }
    }
```

5. Сформувати вікно alert() , у якому повідомлення буде складатися із постійної і змінної частин.
6. Оформіть події Click для кнопок Показати і Сбросити обрані.

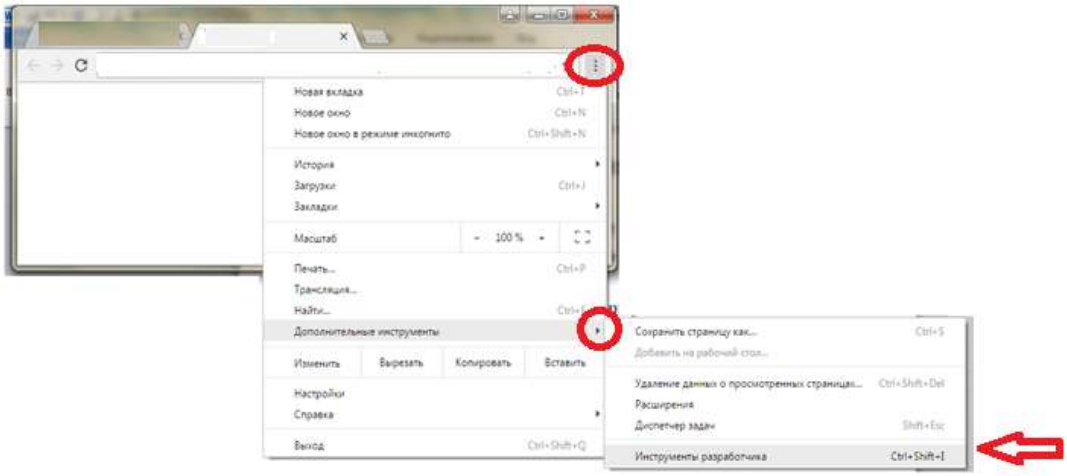
Під час складання і тестування коду використовуйте засоби дебажінгу, які надає браузер.

Інструменти розробника. Web Development Tools

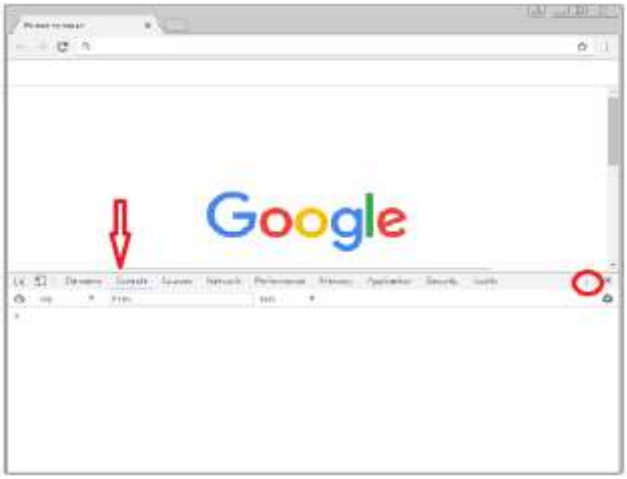
Кожен сучасний інтернет-браузер обладнаний потужними інструментами для web-розробника (Web Development Tools). Ці інструменти дозволяють робити різні речі: від вивчення завантажених зараз HTML, CSS і JavaScript, до відображення яких ресурсів потребує сторінка і як довго вона буде завантажуватися. До того жоден розробник не уникає помилок при розробці коду. На професійному сленгі помилки називають «баги» (від англ. bug), тому процес виправлення помилок називають «дебажінг коду» (debug – отладка). Помилки бувають синтаксичні і логічні. З логічними помилками справа обстоїть досить складно – їх повинен знаходити сам розробник, або тестувальник. А ось синтаксичні помилки (незакриту дужку, не визначену змінну чи функцію тощо) досить вдало знаходить браузер, про що і повідомляє у своїй консолі (Console). Ми вже бачили цей інструмент, коли у JS-коді використовували оператор console.log().

Інструменти розробника досить схожі у різних браузерах. Розглянемо базові функції інструментів розробника в браузері Google Chrome.

Для завантаження можна натиснути комбінацію клавіш Ctrl + Shift + I , або скористатися меню



І ви побачите:



Скориставшись меню цього інструмента можна розташовувати вікно інструментів не тільки знизу вікна браузера, а і у вздовж лівої чи правої сторони. Вікно Інструментів має 9 вкладок, на кожній з яких надана корисна для розробника інформація. Детально про роботу з Інструментами можна прочитати https://learn.javascript.ru/debugging-chrome . Наведемо найбільш цікаві моменти для новачка.

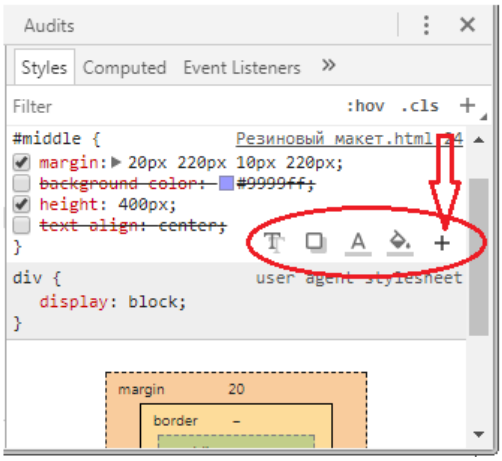
Вкладка Elements дозволяє інспектувати код. Можна виділяти елементи на сторінці, і вони «підганяються» для перегляду коду в Debugger'e. Можна вносити виправлення і дивитися, як це буде виглядати. Однак, при оновленні сторінки внесені нами зміни не фіксуються, і сторінка повертає свій вигляд.
Наприклад,
Відкриємо у браузері html-файл. Візьміть будь-який файл, який ви створювали у розділі Основи CSS. Ми взяли файл «гумової верстки».

Зліва можна виділити елемент, який вас цікавить, а праворуч побачити його стиль і значення чинників Блочної моделі цього елементу. Також праворуч можна редагувати стиль, а у верхній частині вікна браузера побачити, як буде вести себе сторінка із таким стилем.
Приміром ми виділили елемент

```
<div id="middle">"Гумовий" центральний блок</div> == $0
```

і прибрали «флажки» біля обраних правил (скасували фон і вирівнювання за центром)

Якщо ми хочемо додати елементу нове правило слід навести мишею на існуючи правила і у правому нижньому куті побачимо додаткове меню для корегування стилів:



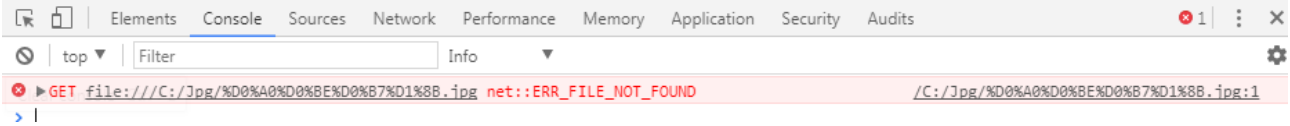
Вкладка Console основний інструмент розробника. Саме на цій вкладці можна побачити помилки і можливість перейти до строки коду, де була виявлена помилка.

Наприклад, ми розробляємо JS-код, який повинен вивести у вікні браузеру картинку, як подію Click на кнопці (увага зараз буде наведено помилковий код):

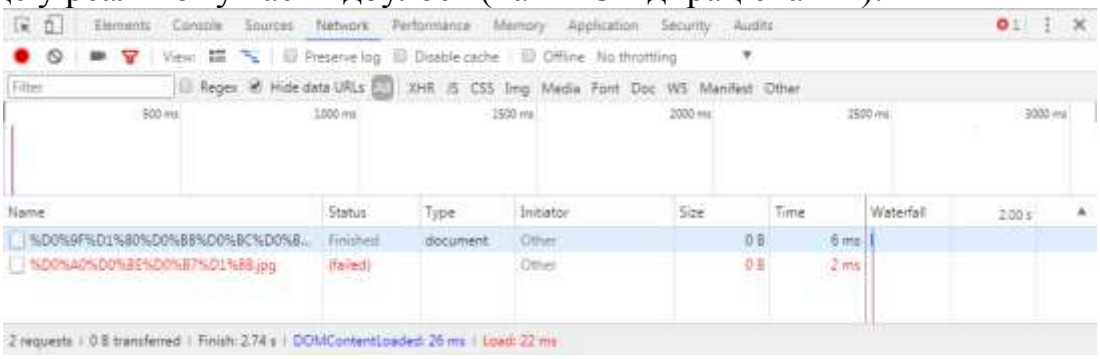
```
<!DOCTYPE html>
<html>
<head>
        <title>Вивчаємо Debbuger</title>
        <meta charset="utf-8">
</head>
<body>
<h2>Для картинки натисни кнопку</h2>
<input type="button" value="Покажі" onclick='showPicture();'>
<script type="text/javascript">
        function showPicture() {
        var img=document.createElement('img');
        img.src='../Jpg/Розы.jpg';
        }
</script>
</body>
</html>
```

Цей код містить дві помилки:

- ✓ ***синтаксичну*** – помилково вказана адреса файлу і браузер не зміг його завантажити (не знайшов, file not found)



Уважно роздивившись повідомлення про помилку, можна побачити і за якою адресою браузер шукав файл Розы.jpg.
На вкладці Network ми теж можемо побачимо який саме файл не знайдений і коли це у реальному часі відбулося (на 2 ms відпрацювання):



Тому у коді слід внести виправлення, вказав повну адресу файлу Розы.jpg:

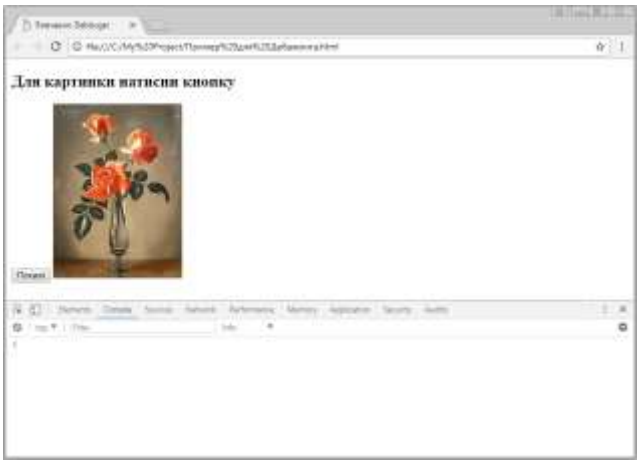
```
img.src='C:/My Project/Jpg/Розы.jpg';
```

Але виправлення теж не приводить до того, що ми побачимо картинку, хоча браузер і перестав повідомляти про помилку.
Якщо Debugger не повідомляє про помилку, а код все одно не видає очікуваного результату – значить у коді є логічні помилки, і розробнику слід ретельно обдумати і сам код і його алгоритм.

- ✓ ***Логічною помилкою*** у нашому прикладі є те, що ми не вказали куди додати створений нами елемент. Тож виправимо ситуацію додав наприкінці тіла функції:

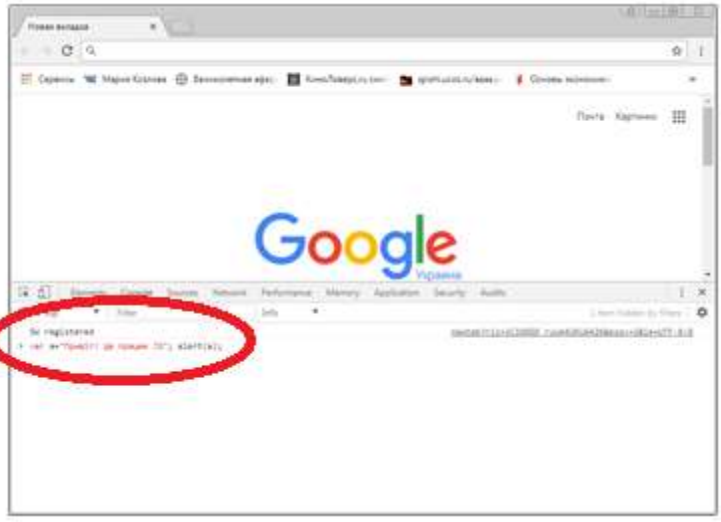
```
document.body.appendChild(img);
```

І побачимо:

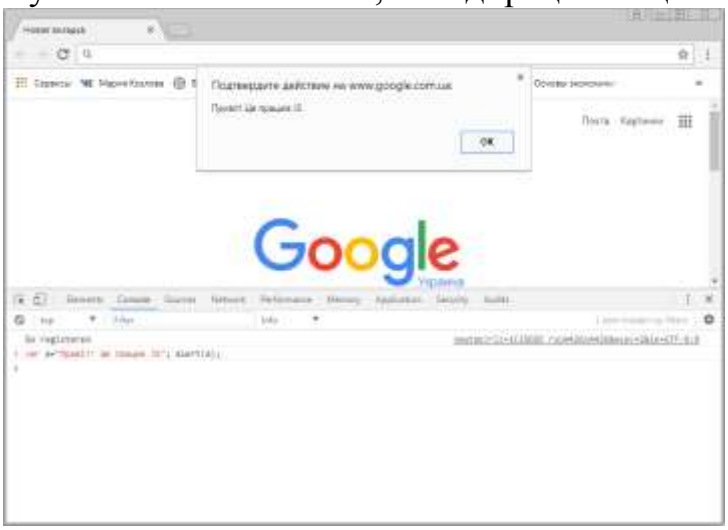


Також на вкладці Console можна записувати фрагменти коду (готового модуля) і натиснувши клавішу Enter побачити результат дії коду.
Наприклад, запишемо у консолі невеличкий приклад:

```
var a="Привіт! Це працює JS"; alert(a);
```



і натиснемо клавішу Enter. Ми побачимо, як відпрацював цей код:



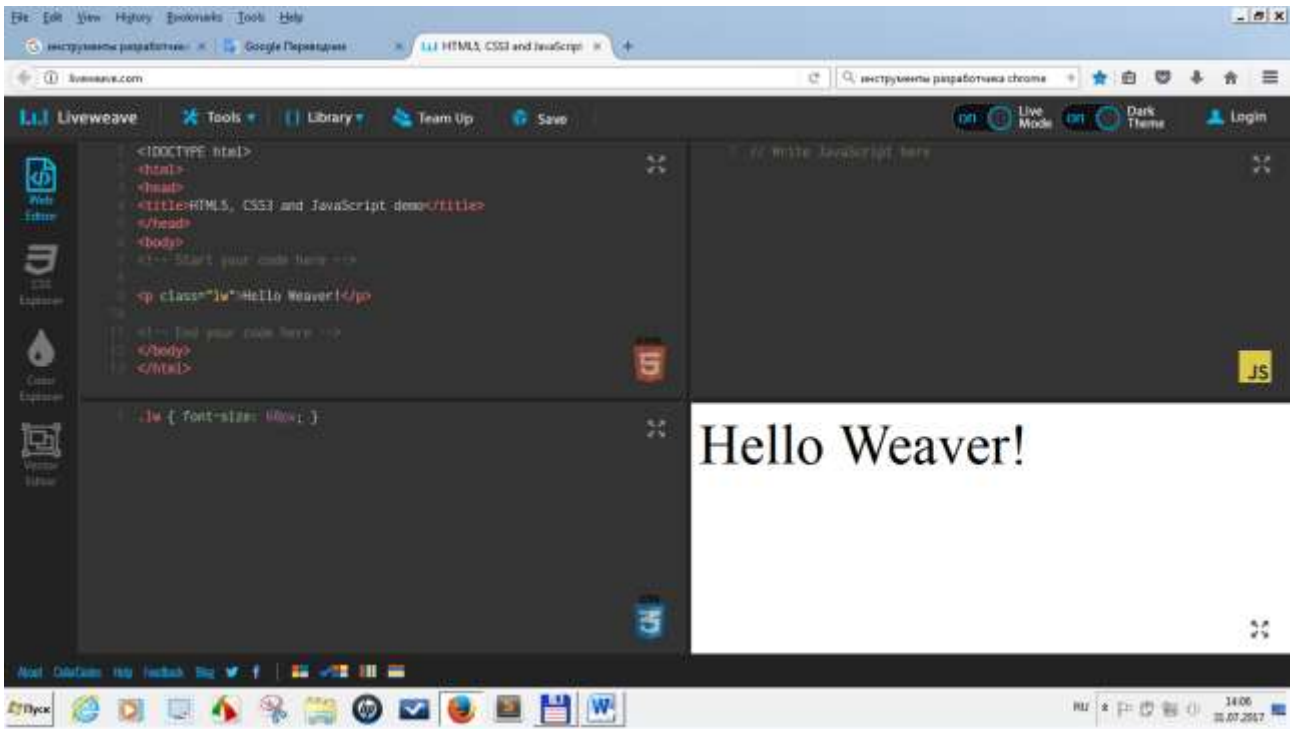
«Дебажіти» код можна і з використанням редактора кодів і перезапускання його у браузері. Для цього, зазвичай, використовуються засоби «проміжного друку», тобто додавання у текст коду тимчасових операторів (після дебагінгу вони видаляються з коду), на кшталт alert() або console.log(), щоб побачити значення об’єктів програмування.

На **вкладці** Sources відбувається власне «дебажінг» (Debugger) коду. Там можна встановлювати «точки останову» (Breakpoint), які дозволяють виконувати код не відразу з початку до кінця, а як би «покроково» (від точки до точки). Коли код зупиниться на Breakpoint, то наводячи мишею на елемент можна побачити всі його значення на даний термін. Продовжити виконання коду можна за допомогою кнопки  (до наступної Breakpoint, або до кінця).

Вкладка Network у реальному часі показує, на які ресурси браузер посилав запит, і які ресурси були завантажені.

Вкладка Timeline показує час, що витрачений на завантаження, рендерінг і повне відображення сторінки у браузері.

Для роботи з окремими фрагментами коду доцільно використовувати on-line редактори. Найбільш поширеним є редактор liveweave, який можна завантажити http://liveweave.com/ , він підтримує сучасні версії HTML 5, CSS 3 і JavaScript. Коди слід записувати у відповідні вікна редактора, не витрачаючи час на оформлення css-стилів і js-скриптів у окремі файли і підключення цих файлів у html-коді. Все, що ви пишете відразу ж відпрацьовує.



Слід пам’ятати, що всі зміни, які ви вносите у код, використовуючи браузер чи on-line редактори, не зберігаються. Тому для збереження проектів використовуйте текстові редактори, або спеціальні off-line редактори коду, наприклад Sublime Text 3, які дають змогу зберегти файли проекту.

Приклад розробки проекту

Процес розробки web- сайту складається з таких етапів:

- планування;
- дизайн;
- розробка;
- підтримка доданку (post-production)

Планування - це

- обговорення із Замовником, формулювання мети створення сайту, вимог до нього, визначення цільової аудиторії майбутніх користувачів;
- розробка дизайну і архітектури сайту;
- розробка документу із специфікаціями;
- розробка прототипу продукту, яку повинен узгодити Замовник.

Дизайн – це верстка сторінок сайту. Етап дизайну теж обов'язково узгодити із Замовником.

Розробка – власне написання функціоналу:

- інтеграція контенту;
- тестування і виправлення помилок («дебажінг»)

Post-production:

- остаточна перевірка Замовником;
- фінальне тестування;
- написання і оформлення супровідної документації;
- пробний запуск продукту;
- отримання відгуків («фідбеків») від користувачів сайту; підтримка

Зазвичай дизайн починається з виготовлення мокапу (англ. mockup) , який є і макетом, і рекламою вмінь розробника майбутнього сайту. Завдяки мокап-файлом можна красиво показати Замовнику, як у реальності буде виглядати дизайн його візитки, сайту, доданку тощо.

Переважна частина мокапів створюється у форматі Photoshop PSD. В Інтернеті існує безліч сайтів, які продають або надають змогу безкоштовно завантажити мокапи. Одним з таких сайтів є сайт https://freebiesbug.com/ , де можна безкоштовно завантажувати мокапи та інші елементи дизайну, *наприклад*, шрифти; колекція сайту постійно поповнюється. Як користуватися завантаженим з Інтернету мокапом створено купу відео курсів у youtube, наприклад, https://www.youtube.com/watch?v=ynRjLUI_1Dw , може знайдете і гарніший ☺ .

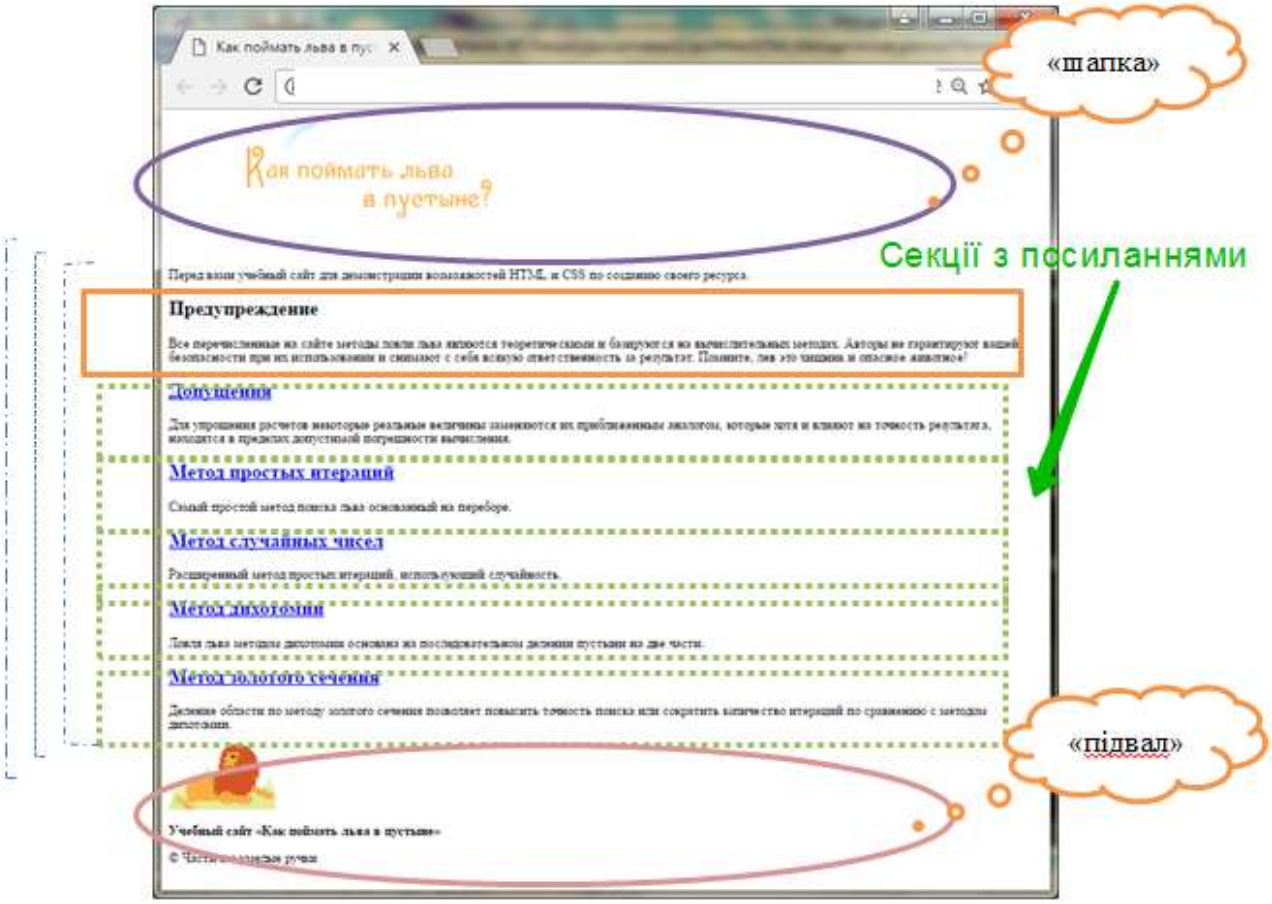
Приклади css правил для типових видів верстки можна переглянути тут http://htmlbook.ru/layout , але не слід забувати, що то є тільки прикладами, які можна й потрібно редагувати, беручі до уваги сучасні рекомендації професіоналів.

Створення сайту «Лев у пустелі»

Даний приклад верстки документу з використання HTML5 і стилів CSS3 було створено за мотивами і з використанням графічних матеріалів прикладу Влада Мержевича, з оригіналом якого можна познайомитися http://htmlbook.ru/samlayout/verstka-na-html5/maket-saita , а сам створений сайт подивитися тут http://lionindesert.ru/ .

Ми розглянемо тільки створення першої (головної) сторінки, в припущенні, що необхідні файли картинок і шрифтів збережені у відповідних папках проекту (images і fonts).

І так, наш макет:



Семантичний макет сторінки такий:

<table>
<tr><td colspan="4">header</td></tr>
<tr><td rowspan="4">div</td><td rowspan="4">div</td><td rowspan="4"></td><td>параграф</td><td rowspan="4"></td></tr>
<tr><td>section (без посилання)</td></tr>
<tr><td>section (з посиланням)</td><td rowspan="3">(всього 5 шт.)</td></tr>
<tr><td>.....</td></tr>
<tr><td>div</td><td>section (з посиланням)</td></tr>
<tr><td colspan="4">footer</td></tr>
</table>

Треба створити html-файл **index.html** додав у нього базову структуру html-коду і вказати кодування <meta charset="utf-8" >. Потім сформувати основну

семантичну структуру. І наприкінці формування html-коду додати класи елементам.

1. **Порядок роботи на малюнку**:

```
<body>
  <header>
    <div class="header-bg">
      <img src="images/header-title.png" alt="Картинка header-title">
    </div>
  </header>
  <div class="content-gradient">
    <div class="content-bg">
      <div class="content-white">
        <p>  ———————— </p>
        <section class="warning">
        <h2>  ———————— </h2>
          <p>  ———————— </p>
        </section>
        <section>
        <h2><a href="#">  ——— </a></h2>
          <p>  ———————— </p>
        </section>

      </div>
    </div>
  </div>
  <footer>
    <div class="lion"><img src="images/lion.png" alt="Тут картинка lion"></div>
    <div class="footer-bg">
      <div class="copyright">
        <p><strong>  ———————————————— </strong></p>
        <p>&copy;  ———————————— </p>
      </div>
    </div>
  </footer>
</body>
```

2 Скопіювати "секцію" 4 рази и виправити текст у "нових секціях"

3 Додати класи елементам

1 Додати відповідний текст

Зверніть увагу, у посиланні стоїть хештег («#»), який можна завжди змінити на реальне посилання.

Зверстана сторінка буде мати вигляд:

2. **Оформимо сторінку** за допомогою стилів CSS 3. Файл стилю style.css у кодуванні UTF-8 збережемо у папці CSS.

У html- коді сторінки у елемент head додамо елемент link для підключення файлу стилів до сторінки:

<link rel="stylesheet" type=”text/css” href="CSS/style.css">

Додавати стилі будемо покроково, контролюючи їх застосування (відображаємо сторінку у браузері):

2.1 Приберемо **поля на сторінці**. Призначимо **тип шрифту і його розмір** для всіх елементів сторінки:

```
body {
margin: 0;            /*всі  поля */
font-family:  Arial, Helvetica, sans-serif;
font-size: 20px;
}
```

2.2 **Всі семантичні елементы** вказуємо, як блочні («**Веди себе, як блок**»):

```
header, section, footer {
display: block; /* Блочный элемент */
}
```

2.3 **оформлюємо «шапку»**. Селекторами є тег header і імена класів

```
header {
/* градієнтна заливка фона  */
  background: linear-gradient(#00B0D8,#FFFFFF);
}
```

```
.header-bg {
/*фонова картинка , повторення по X, розташування знизу по центру*/
  background: url(../images/header-animal.png) repeat-x center bottom;
  height: 405px;              /* висота шапки */
  text-align: center;                 /* вирівнювання по центру */
}

.header-bg img {
  position: relative;                 /* відносне позиціювання */
  top: 40px;                  /* зрушуємо картинку до низу */
  width: 456px;               /* розміри картинки */
  height: 166px;
}
```

2.4 Оформлюємо основну частину. Селектори імена відповідних класів елементів div**:**

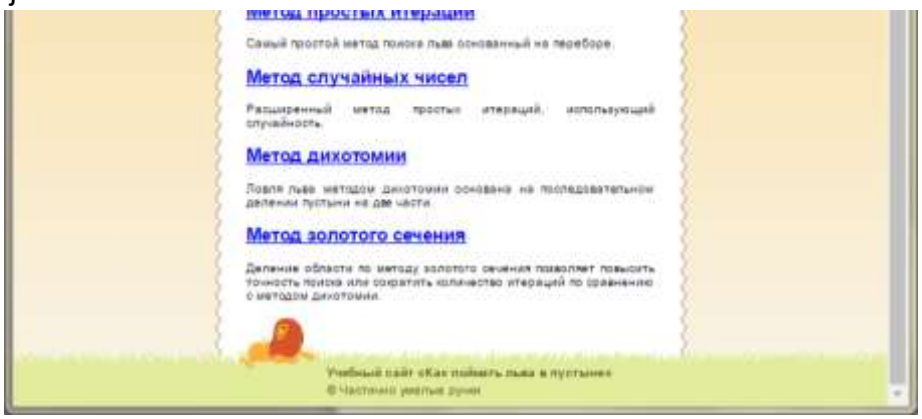
```
.content-gradient {
  background: linear-gradient(#f9db94,#f9f2e3);
}

.content-bg {
  width: 760px;       /* ширина макету */
  margin: auto;       /* вирівнювання блоку по центру вікна браузеру*/
/* додаємо фонову картинку з повторенням вздовж Y */
  background: url(../images/content-bg.png)  repeat-y;
}

.content-white {
  background: #ffffff;       /* білий колір фону */
  margin: 0px 11px;          /* поля зверху і знизу */
```

```
  padding: 20px 40px 80px;      /* внутрішні відступи */
  text-align: justify;          /* вирівнювання по ширині */
}
```

2.5 Оформлюємо «підвал»:

```
footer {
/* Фоновий малюнок трави , без повторів*/
  background: url(../images/grass.png) 50% 53px no-repeat;
  margin-top: -77px;            /* поля зверху */
  overflow: auto;               /* скасовуємо відступи */
  position: relative;           /* відносне позиціювання */
}

.lion {
  position: absolute;           /* абсолютне позиціювання */
  top: 3px;                     /* зверху від вікна браузеру */
  left: 50%;                    /* буде по центру */
  margin-left: -347px;          /* поля зліва */
}

.lion img {
  width: 130px;             /* розміри картинки */
  height: 80px;
}

.footer-bg {
  background: #e2ed9c;          /* фоновий колір */
  margin-top: 80px;             /* поля зверху */
}
```

```
.copyright {
  width: 740px;                    /* ширина макету без відступів */
  padding: 0 10px 10px;            /* відступи */
  margin: auto;                 /* вирівнювання по центру вікна браузеру */
  color: #526118;                  /* колір тексту */
}

.copyright p {
  margin: 0 0 5px 170px;           /* поля  блоку*/
}
```



2.6 Оформлюємо секцію з «попередженням»

```
.warning {
  overflow: hidden;          /* відміняємо обтікання */
  margin: 30px 0;            /* поля зверху і знизу */
/* фонова картинка без півторення  */
  background: url(../images/head.png)  no-repeat;    min-height: 92px;
}

.warning h2, .warning p {
  margin: 0 0 0 70px;        /* поля */
}
```

2.7 Оформлюємо секції з посиланнями на «інші документи»

```
a {
  color: #1b75bc;           /* колір посилань */
  text-decoration: none;  /* прибрати підкреслення тексту в посиланнях */
}

/* буде відбуватися зміна коліру при наведенні на посилання */
a:hover {
  color: #d6562b;
}

section h2 {
  font-size: 25px;          /* розмір тексту */
  font-weight: bold;        /* жирний текст */
  margin-bottom: 0;                 /* поле знизу */
}
```

Сайт створено ☺

Створення сайту «Колекція зображень і відео»

Створимо динамічну сторінку за макетом

Сторінка вміщує панель навігації із «заготовками» посилань та формою реєстрації (зверху) і шість однотипних блоків з картинкою (або відео) і описом (невеличкий текст). Особливістю є те, що текст опису цілком не видно у наданій області, тому буде додана функціональність (скріпт) – кліком миші по відповідній кнопці під текстом опис «розгортається» так, що становиться видимим цілком, або «згортається» до вихідного розміру.

Всі тексти для розробки (тестування) цього проекту будуть взяті за допомогою генератора текстів з сайту http://ru.lipsum.com/. Тексти не несуть смислового навантаження,а виконують роль «заповнювача».

Вже за звичною для вас традицією розробку проекту будемо починати зі створення html-коду розмітки мовою HTML5, до якої додамо стилів CSS3 і скріпт мовою JavaScript.

Почнемо зі створення html-коду.

Проаналізуємо загальну структуру html-файлу:

Документ складається з 4 семантичних частин:

1. «Шапка» сторінки. Для її розмітки будемо використовувати елемент <header>.
2. Головне меню (навігація), яка містить гіперпосилання і login форму. Головне меню розмітимо тегом <nav>.
3. Основна частина документу з картинками і відео буде розмічена тегом <main>
4. «Підвал» документу оформимо у тегі <footer>.

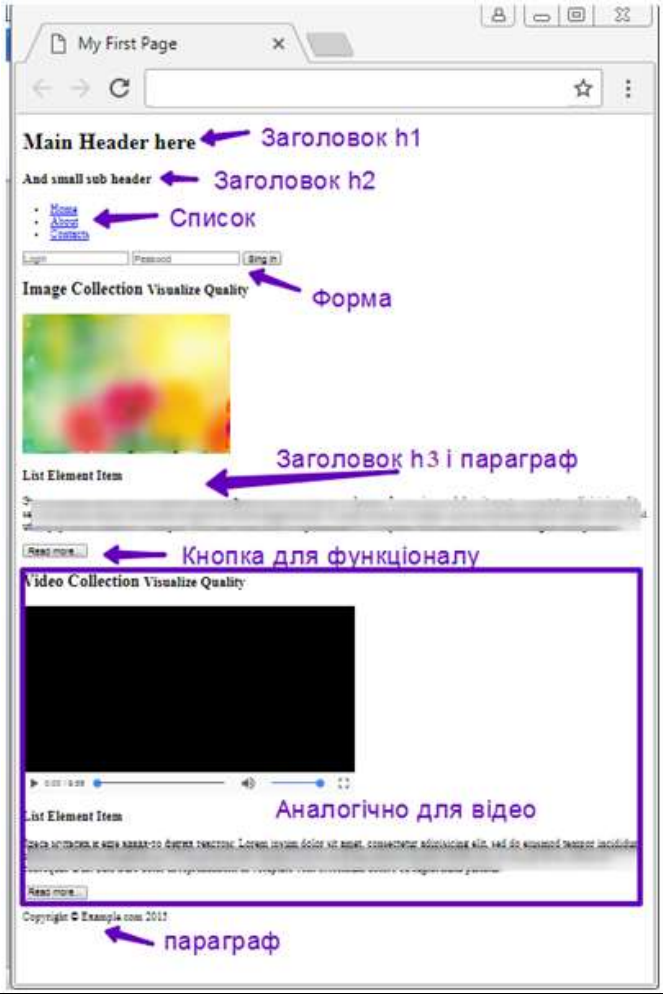
Таким чином «каркас» документу такий:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Collection</title>
        <meta charset="utf-8" />
    </head>
    <body>
        <header>
            <!-- Тут буде "шапка" документу -->
        </header>
        <nav>
            <!-- Тут буде головне меню з login формою  -->
        </nav>
        <main>
            <!-- Тут буде основна частина документу -->
        </main>
        <footer>
            <!-- Тут буде "підвал" документу -->
        </footer>
    </body>
</html>
```

Почнемо наповнювати вмістом кожну з складових частин «каркасу». **Header складається** з заголовка (тег h1) и підзаголовка (тег h2). Додатково «завернемо» внутрішню частину header'а у елемент . «Загортати» у тег будемо і вміст решти складових «каркасу», в подальшому це буде використано при додаванні стилів документу.

Для footer'а , очевидно, потрібен параграф <p> , але і його «завернемо» у тег .

Створемо «найпростіші» частини **header і footer**:




```
<header>
  <div>
     <h1>Назва сайту</h1>
     <h2>
      <small>Підзаголовок</small>
     </h2>
  </div>
</header>
```

```
<footer>
  <div>
    <p>Copyright © Example 2017</p>
  </div>
</footer>
```

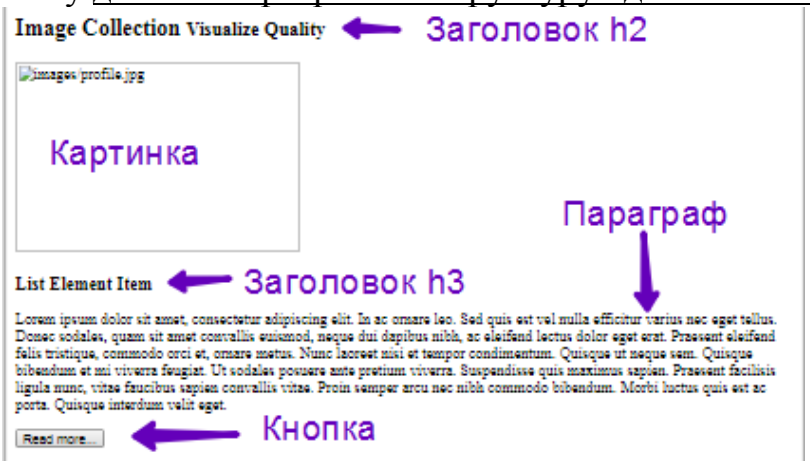
Вміст Головного меню (навігації) складається з маркованого списку, елементами якого є гіперпосилання. Далі розташована login форма. **Форма складається** з двох полів і кнопки відправки форми.

```
<nav>
        <div>
                <ul>
                        <li><a href="#">Home</a></li>
                        <li><a href="#">About</a></li>
                        <li><a href="#">Contacts</a></li>
                </ul>
                <form>
                        <input  type="text" placeholder="Login" required>
                        <input  type="passwod" placeholder="Passwod" required>
                        <input  type="submit" value="Sing in">
                </form>
        </div>
</nav>
```

Атрибут **placeholder** елементу input надає текст всередині текстового поля, який зникає при отриманні елементом фокуса. Атрибут **required** забезпечує те, що на сервер не буде відправлена пуста (незаповнена) форма. При спробі надіслати пусту форму з'явиться повідомлення про помилку.

Проаналізуємо **склад основної частини** і розробимо методику її створення. Основна частина складається з двох однотипних фрагментів, кожен з яких вміщує по три однакові за структурою елементи

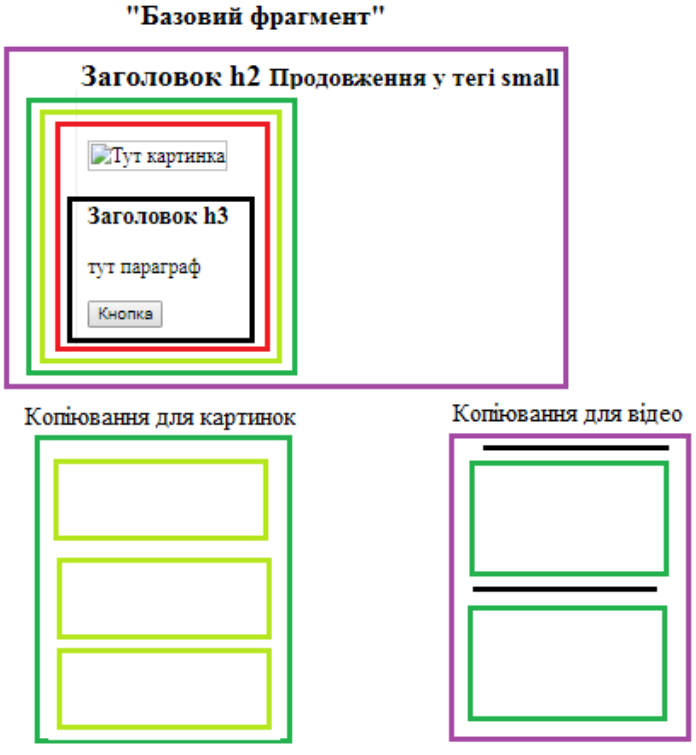
«Закресленими» надано блоки, які будуть отримані копіюванням і наступним редагуванням. Тому достатньо розробити структуру одного типового блоку:



Перед написанням коду уважно передивіться структуру вкладених елементів (нижче вони надані у прямокутниках):

```
<main>
        <div>
                <h2>Заголовок h2  <small>Продовження у тегі small</small></h2>
                <div>
                        <div>
                                <div>
                                        <img src="#" alt="Тут картинка" >
                                        <div>
                                                <h3>Заголовок h3</h3>
                                                <p>Тут параграф</p>
                                                <input type="button" value="Далі ...">
                                        </div>
                                </div>
                        </div>

<!--Сюди скопіюємо фрагмент  два рази , щоб отримати колекцію картинок-->

                </div>

<!--Сюди скопіюємо фрагмент  для відео колекції -->

        </div>
</main>
```

На схемі наведено загальну структуру блоків :



Копіювання варто проводити на заключному етапі формування сторінки.

Для створення зручних (при додаванні стилів) селекторів,
додайте в атрибути елементів класи:

Семантичним тегам надайте клас з ім'ям тегу:

```
<header class="Header">
<nav class="Navigation">
<form  class="Login">
<footer class="Footer">
```

Кожному першому div елементу у блоках header , nav , main і footer надайте клас Container:

```
<div class="Container">
```

Решті елементів div у елементі main надамо такі класи:

```
<div class="Collections">
    <div class="Collection-item-outer">
        <div class="Collection-item">

            …….

            <div class="Collection-text">

                ……
```

Елементу h2, який міститься у main, надамо клас Collection-title

```
<h2 class="Collection-title">Заголовок h2 <small>Продовження у тегі small</small></h2>
```

Створимо у папці css файл styleCol.css і додамо його у head за допомогою link

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

Вміст файлу стилів:

```
/* GENERAL STYLES */
body {
    font-family: Arial, san-serif;
    font-size: 100%;
}
.Container {
    margin: 0 auto;
    width: 960px;
}
ul {
    margin: 0;
```

```
        padding: 0;
}
ul li {
        list-style-type: none;
}
a {
        text-decoration: none;
}
p {
        font-size: 13px;
        line-height: 1.3em;
}
/* HEADER */
.Header {
        padding: 50px 0px;
        background-color: yellowgreen;
}
.Header h1,
.Header h2 {
        color: greenyellow;
        text-align: center;
}
/* NAVIGATION */
.Navigation {
        background-color: grey;
        border-color: #101010;
        min-height: 64px;
}
.Navigation li {
        float: left;
        width: 100px;
}
.Navigation a {
        display: block;
        color: #9d9d9d;
        line-height: 4em;
        font-weight: bold;
        text-align: center;
}
.Navigation li a:hover {
```

```
        color: yellowgreen;
        background-color: black;
}
/* LOGIN FORM */
.Login {
        padding: 15px 0;
        float: right;
}
.Login input[type="text"],
.Login input[type="password"] {
padding: 5px;
margin-right: 5px;
border: 1px solid transparent;  /* прозорий колір */
border-radius: 5px;
outline: none;
}
.Login input[type="submit"] {
        padding: 5px 15px;
        border: 1px solid lightgrey;
        border-radius: 5px
}
.Login input[type="submit"]:hover,
input[type="button"]:hover {
        background-color: darkgrey;
        color: white;
}

/* MAIN PART */
.Collections {
        overflow: auto;
}
.Collection-title {
        padding-bottom: 5px;
        border-bottom: 1px solid #777;
        font-size: 36px;
}
.Collection-title small {
        color: #777;
        font-size: 70%;
}
```

```
.Collection-item-outer {
        display: inline-block;
        float: left;
        padding-left: 5px;
        padding-right: 5px;
        width: 33.33%;       /* три  «гумових» блока  */
        box-sizing: border-box;
}
/* Вирівнювання границі у  "Коллекції картинок" */
.Collections > div:first-child {
        padding-left: 0;
        padding-right: 5px;
}
.Collections > div:last-child {
        padding-right: 0;
        padding-left: 5px;
}
/* Кінець правил вирівнювання */
.Collection-item {
        border: 1px solid #959595;
}
.Collection-item img,
.Collection-item video {
        width: 308px;
        height: 300px;
}
.Collection-text {
        padding: 0 20px 20px 20px;
}
.Collection-text h3 {
        color: yellowgreen;
        overflow: hidden;
        white-space: nowrap;     /* заборона реакції на пробіли */
/* додавання наприкінці три крапки, якщо весь текст не помістився*/
        text-overflow: ellipsis;
}
.Collection-text p {
height: 80px;
overflow: hidden;           /* сховати все, що перебільшує 80px */
}
.Collection-text input[type="button"] {
```

```
        padding: 10px;
        border: 1px solid transparent;
        border-radius: 5px;
        outline: none;
}
/* FOOTER */
.Footer {
        background-color: yellowgreen;
        color: white;
        padding: 20px 0;
        margin-top: 30px;
}
```

Додамо нашій сторінці інтерактивності.

В «Описі картинки» текст, який міститься у елементі <p>, перевищує надану у стилі висоту 80px для цього елементу, і тому текст цілком не видно (текст «прихований»). Для тестування скріпту згенеруйте у цей параграф «заповнювач» тексту з 100 слів.

Напишемо скріпт, який «розгортає» весь текст, якщо він «прихований», або навпаки «приховує» решту тексту. Скріпт зв'яжемо із подією Click по кнопці під текстом. Для того, щоб користувачеві було зрозуміло, на кнопці буде напис «Далі…», якщо текст «прихований», і напис «Згорнути», якщо текст був видимий цілком.

У папці JS створимо файл скріпту scriptCol.js.

У html-коді додамо у елемент head посилання на файл із скріптом:

```
<script type="text/javascript" src="JS/scriptCol.js"></script>
```

У атрибутах елементу, подія якого буде оброблятися за допомогою скріпту, запишемо:

```
<input type="button" value="Далі ..." onclick="showText(this);">
```

Тепер у файлі scriptCol.js напишемо функцію:

```
function showText(el) {
        // перевіримо висоту элемента над "кнопкою"
if (el.previousElementSibling.clientHeight === 80) {
        el.previousElementSibling.style.height ="100%";
        el.value="Згорнути";
  } else {
        el.previousElementSibling.style.height="80px";
        el.value="Далі...";
    }
```

}

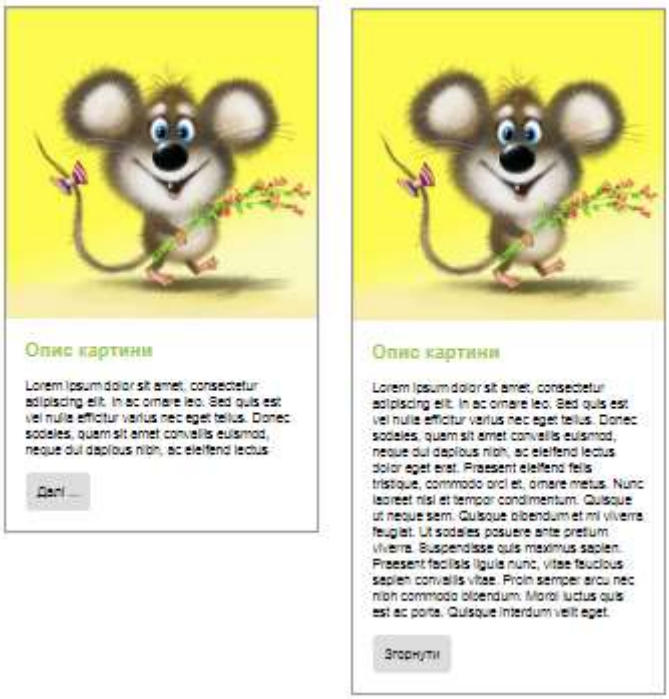
У якості фактичного параметру функції showText(el) ми використали «*об'єкт прив'язки*» **this** – це ключове слово спеціального аргументу, який дає змогу звернутися до елементу «кнопка» (з якої ми визвали функцію);

у функції використовувався метод **previousElementSibling** , який надає можливість «доступитися до сусіда зверху»;

clientHeight - властивість, яка доступна тільки на читання, тобто надає змогу отримати висоту елементу.

Для того, щоб встановити висоту елементу ми використали властивість **style.height**.

Протестуємо роботу скріпту і побачимо:



Тепер саме час закінчити створення проекту, скопіював і відредагував блоки. В скопіюємо два рази . Отримаємо три блоки картинок.

Тепер в скопіюємо один раз весь його вміст, починаючи з <h2 class="Collection-title">Картини <small>Високої якості</small></h2> , і замінимо тут контент «Картини» на контент «Відео».

Для того, щоб в нас з'явилися блоки з відео, треба елемент, який розмічує картинку, замінити на елемент, який розмічує відео, а саме:

замість

```
<img src="#" alt="Тут картинка">
```

записати

```
<video controls>
    <source src="#" type="video/mp4">
</video>
```

Зверніть увагу, що у якості відео слід використовувати файли mp4 , які розміщуємо у папці video.

Атрибут controls доданий у тег <video> додасть до відео панель управління відображенням:

Словник термінів

Back-end (ще можете зустрічати як **server-side**) - все, що відбувається за лаштунками web-додатків. Back-end переважно використовує базу даних для збереження інформації, за допомогою якої потім генерується front-end.

CDN (Content Delivery Network). Простими словами – це багато серверів, розкиданих по світу, які містять одну і ту ж інформацію.

CMS, від англійського Content Management System (система керування контентом), - це програмне забезпечення, що дозволяє користувачам розміщувати або змінювати вже розміщену на сайті інформацію без залучення розробників сайту. Це значить, що користувачеві не обов'язково мати навички програмування або знання мови HTML, щоб, наприклад, опублікувати на своєму сайті новину, статтю або додати зображення. Часто поряд із терміном CMS можна почути також термін «**движок сайту**», яким звичайно користуються web-майстри у своєму професійному сленгу.

CSS (Cascading Style Sheets) – код, який описує правила стилю для відображення web-сторінок. Це не мова програмування, а лише таблиця стилів. CSS «вирішує» як саме буде виглядати та де розміщуватись ваш HTML код. CSS синтаксис – це правила написання CSS.

DNS-сервери (Domain Name System) зберігають інформацію про вузли, імена яких належать домену і виконують трансляцію їх символьних імен в числові IP-адреси наприклад, імені Wikipedia.org відповідає адреса 91.198.174.192. Кожний домен має унікальне ім'я, а кожен комп'ютер, підключений до Інтернету, має, як правило, доменне ім'я. Домени мають між собою ієрархічні відношення.

DOM (Document Object Model, Обєктна модель документа) – структура документа, представлена у вигляді дерева.

ECMA (European Computer Manufacturers Association) – некомерційна асоціація європейських виробників комп'ютерів, створена в 1961 році і розташована в Женеві. Займається розробкою стандартів для інформаційних і комунікаційних систем.

Front-end (ще можете зустрічати як **client-side**) - все, що бачить користувач, коли завантажує web-сторінку в браузері і, власне, все, що може показати та виконати браузер.

JavaScript – скріптова мова програмування, яка широко використовується для додавання функціональності та інтерактивності web-сторінкам.

jQuery – бібліотека мови JavaScript. Бібліотека мови програмування - збірка об'єктів чи підпрограм для вирішення близьких за тематикою задач.

HTML (Hyper Text Markup Language) – спеціальний код, призначений для розмітки web сторінок.

IDE (Integrated Development Environment) – це редактор коду, який має більше можливостей, може працювати з допоміжними системами, такими як багтрекер, контроль версій і т.д.

Node.js – платформа з відкритим сорс-кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Платформа node.js перетворила мову JavaScript з переважно клієнтським використанням у браузерах на мову загального використання з великою спільнотою розробників.

Scope (**область видимості**) – набір змінних, об'єктів і функцій, до яких є доступ з конкретної частини програми.

SEO (Search Engines Optimization) – оптимізація сайту в пошукових системах, тобто, проведення заходів з просування сайту на верхні позиції в результатах пошуку.

URL (Uniform Resource Locator) – визначник місцезнаходження сайту в мережі Інтернет. URL складається з домену, шляху до сторінці та імені її файлу. Був винайдений в 1990р. Тімом Бернерс-Лі. Спочатку URL виступав як адреса розташування файлів. На сьогоднішній момент застосовується практично для всіх існуючих ресурсів в мережі Інтернет, позиціонуючись як частина більш загальної системи ідентифікації сайтів – **URI** (International Resource Identifier).

Web програмування або **web розробка** – це створення динамічних web додатків та web сайтів.

Адаптивна розмітка, адаптивний дизайн – розмітка, яка змінюється залежно від ширини робочої області вікна браузера.

Баг (bug)- означає помилку, ваду або дефект в програмі або системі, що викликає в ній неправильний або неочікуваний результат або неочікувану поведінку

Багтрекер (bug tracker) - прикладна програма, розроблена, щоб допомогти тестерам та програмістам відстежувати історію звітів про баги під час своєї роботи.

Бази даних - сукупність відомостей, об'єднаних за певною ознакою. Головне завдання бази даних — гарантоване збереження значних обсягів інформації (так звані записи даних) та надання доступу до неї користувачеві або ж прикладній програмі.

Бібліотека – набір допоміжних функцій, які вирішують одну конкретну задачу. Бібліотека може додавати певний функціонал у вашу програму, але Ваш код залишається тим же Вашим кодом.

Браузер - це програма, що дозволяє відображати сторінки мережі Інтернет на екрані комп'ютера, передавати і отримувати дані з «всесвітньої павутини».

Валідатор – програма, яка перевіряє правильність написання програмного коду і його відповідність стандартам.

Віджет – це графічний модуль , а точніше невеличкі програмні доданки, які знаходяться на Робочому столі і можуть бути постійно використані для перегляду погоди, часу, новин, пошти, гри у яку-небудь невеличку просту гру, а також використовуються для швидкої передачі даних без допомоги web-браузера.

Доме́нне ім'я (*англ*. Domain name) або **Доме́н** (англ. Domain, від лат. dominium — володіння) — частина простору ієрархічних імен мережі Інтернет, що обслуговується групою серверів системи доме́нних імен (DNS-серверів) та централізовано адмініструється.

Елемент – основні компоненти мови розмітки HTML. HTML-документ складається з головного елементу html, до змісту якого додаються інші елементи.

Модуль - функціонально завершений фрагмент комп'ютерної програми.

Модульне тестування, або **юніт-тестування**, або **unit-testing** - це частина технологічного процесу розробки програмного проекту, що дозволяє перевірити на коректність окремі модулі вихідного коду програми.
У більшості випадків модульні тести створюються розробниками програм. Їх написанням займається автор коду, що підлягає тестуванню, або співробітники тієї ж робочої групи, в яку входить автор коду. Який елемент програми слід вважати модулем, який треба тестувати, - вибирає розробник тестів. Як правило, модулем вважається окрема функція (метод) програми.

Мокап (Mock up або макет) – модель певного об'єкта в спрощеному вигляді для того, щоб скласти загальне уявлення про нього.

Редактор коду – програма, створена спеціально для написання коду на будь-якій мові програмування.

Рефакторинг – це контрольований процес покращення вашого коду, без написання нової функціональності.

Селектор – дозволяє звернутись до конкретного HTML елемента або кількох HTML елементів.

Семантика – зміст, логічне наповнення HTML елемента.

Сервер – це комп'ютер, який обслуговує всі підключені до нього персональні комп'ютери. При цьому сервер виконує свої функції і завдання без участі в цьому процесі людини, тобто самостійно.

Середовище розробки - це комп'ютерна програма, що допомагає розробнику створювати нове програмне забезпечення чи модифікувати (удосконалювати) вже існуюче.

Скріпт – набір команд на тій чи іншій мові програмування.

Сорс-код – визначає програму, яка, зазвичай, міститься в одному або більше текстових файлах, іноді зберігається в базах даних, як збережені процедури, а також може з'явитися, як фрагменти коду, надруковані в книжках або інших засобах друку.

Тег (tag) – мітка, яку ви використовуєте для вказівки браузеру, як він повинен показувати ваш web-сайт.

Функція – блок коду, який визначається один раз і може викликатись багаторазово.

Фреймворк – це своєрідний каркас для створення комп’ютерних програм. Фреймворк «нав’язує» розробнику певні архітектурні обмеження при створенні програм.

Хо́стинг (*англ.* hosting) — послуга, що включає надання дискового простору, підключення до мережі та інших ресурсів для розміщення фізичної інформації на сервері, що постійно перебуває в мережі (наприклад Internet). Поняття хостингу включає в себе широкий спектр послуг із використанням різного апаратного та програмного забезпечення. Зазвичай під поняттям послуги хостингу мають на увазі, як мінімум, послугу розміщення файлів сайту на сервері, на якому запущене програмне забезпечення, необхідне для обробки запитів до цих файлів (web-сервер).

Корисні посилання

1. Самоучитель HTML [Електронний ресурс] – Режим доступу до ресурсу: http://htmlbook.ru/samhtml
2. HTML5 [Електронний ресурс] – Режим доступу до ресурсу: http://htmlbook.ru/html5.
3. Справочник HTML [Електронний ресурс] – Режим доступу до ресурсу: https://webref.ru/html.
4. Магомедов З. Основы семантической верстки на HTML5 [Електронний ресурс] / Заур Магомедов. – 2015. – Режим доступу до ресурсу: https://zaurmag.ru/priemy-verstki-html-css/osnovy-semanticheskoj-verstki-na-html5.html.
5. Самоучитель CSS [Електронний ресурс] – Режим доступу до ресурсу: http://htmlbook.ru/samcss.
6. Справочник CSS [Електронний ресурс] – Режим доступу до ресурсу: https://webref.ru/css.
7. Язык JavaScript: история возникновения, версии и стандарты [Електронний ресурс] // OwlWeb.ru - блог о продвижении и создании сайтов. – 2016. – Режим доступу до ресурсу: http://owlweb.ru/yazyk-javascript-istoriya-vozniknoveniya-versii-i-standarty/.
8. Ruslan. Версії CSS про переваги CSS і трохи історії [Електронний ресурс] / Ruslan // ZURA-BLOG. – 2015. – Режим доступу до ресурсу: http://ruszura.in.ua/korysni-prohramy-i-servisy/versiji-css-pro-perevahy-css-i-trohy-istoriji.html.
9. Современный учебник Javascript [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/.
10. Изменение страницы посредством DOM [Електронний ресурс] – Режим доступу до ресурсу: http://javascript.ru/tutorial/dom/modify.
11. Функции [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/function-basics.
12. Советы по стилю кода [Електронний ресурс] – Режим доступу до ресурсу: http://learn.javascript.ru/coding-style.
13. Кантор И. Открытие окон и методы window [Електронний ресурс] / Илья Кантор. – 2017. – Режим доступу до ресурсу: https://learn.javascript.ru/window-methods.
14. Дудин Д. Что такое Flexbox? Описание всех css свойств, основные принципы, преимущества и недостатки. [Електронний ресурс] / Дмитрий Дудин. – 2014. – Режим доступу до ресурсу: http://html5.by/blog/flexbox/.
15. alexriz. Практическое применение FlexBox [Електронний ресурс] / alexriz // Хабрахабр. – 2014. – Режим доступу до ресурсу: https://habrahabr.ru/post/242545/.

16. Гудок А. DIV верстка резиновых блоков в разметке html/css float [Електронний ресурс] / Александр Гудок. – 2010. – Режим доступу до ресурсу: http://skillcoding.com/Default.aspx?id=204.
17. Український громадський проект масових відкритих онлайн-курсів «Prometheus». [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://prometheus.org.ua/.
18. Что такое UX и UI дизайн – особенности и отличия [Електронний ресурс] – Режим доступу до ресурсу: https://www.kasper.by/blog/chto-takoe-ux-i-ui-dizain/.
19. Веб программирование для начинающих. Как сделать сайт самому [Електронний ресурс] // CodeAcademy. – 2013. – Режим доступу до ресурсу: http://codeacademy.ru/?page=1.
20. Веб-програмування. jQuery у веб-додатках [Електронний ресурс] // студія веб-дизайну WebStudio2U. – 2015. – Режим доступу до ресурсу: http://webstudio2u.net/ua/programming/120-jquery.html.
21. Уроки jQuery [Електронний ресурс] – Режим доступу до ресурсу: https://www.site-do.ru/js/jquery1.php.
22. Обзор инструментов разработки в браузерах [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Discover_browser_developer_tools.
23. Яровая М. 17 полезных Chrome-расширений для дизайнеров и веб-разработчиков [Електронний ресурс] / Майя Яровая. – 2016. – Режим доступу до ресурсу: https://ain.ua/2016/09/26/17-poleznyx-chrome-rasshirenij-dlya-dizajnerov-i-veb-razrabotchikov.
24. HTML [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/HTML.
25. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org.
26. Way J. The 30 CSS Selectors You Must Memorize [Електронний ресурс] / Jeffrey Way. – 2011. – Режим доступу до ресурсу: https://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048.
27. Lorem Ipsum [Електронний ресурс] – Режим доступу до ресурсу: http://ru.lipsum.com/.
28. XnView MP Програма для зменшення кількох фото [Електронний ресурс] – Режим доступу до ресурсу: http://www.xnview.com/en/xnviewmp/. (відео-урок https://zaurmag.ru/videouroki/kak-odnim-razom-umenshit-razmer-neskolkih-foto.html)

Таблиця імен
і відповідних шістнадцятерічних RGB кодів для кольорів

У CSS і JavaScript можна використовувати як in English, та і code RGB, але перед останнім слід додавати хештег «#», *наприклад*, #fffafa

In English	Code RGB	In English	Code RGB
snow	fffafa	spring green	00ff7f
ghostwhite	f8f8ff	lawn green	7cfc00
antique white	faebd7	green (*)	00ff00
cream	fffbf0	medium green	c0dcc0
peachpuff	ffdab9	dark green	008000
navajo white	ffdead	chartreuse	7fff00
cornsilk	fff8dc	green yellow	adff2f
ivory	fffff0	lime green	32cd32
lemon chiffon	fffacd	yellow green	9acd32
seashell	fff5ee	forest green	228b22
honeydew	f0fff0	forest green	f0e68c
azure	f0ffff	pale goldenrod	eee8aa
lavender	e6e6fa	light goldenrod yellow	fafad2
lavender blush	fff0f5	light yellow	ffffe0
misty rose	ffe4e1	yellow (*)	ffff00
white (*)	ffffff	dark yellow	808000
black (*)	000000	gold	ffd700
dim gray	696969	light goldenrod	ffec8b
slate gray	708090	goldenrod	daa520
light slate gray	778899	burly wood	deb887
gray	bebebe	rosy brown	bc8f8f
light gray	c0c0c0	saddle brown	8b4513
medium gray	a0a0a4	sienna	a0522d
dark gray	808080	beige	f5f5dc
midnight blue	191970	wheat	f5deb3
navy (*), dark blue	000080	tan	d2b48c

cornflower	6495ed	chocolate	d2691e
slate blue	6a5acd	firebrick	b22222
light slate blue	8470ff	brown	a52a2a
royal blue	4169e1	salmon	fa8072
blue	0000ff	light salmon	ffa07a
sky blue	87ceeb	orange	ffa500
light sky blue	87cefa	coral	ff7f50
steel blue	4682b4	light coral	f08080
light steel blue	b0c4de	orange red	ff4500
light blue	a6caf0	red (*)	ff0000
powder blue	b0e0e6	dark red	800000
pale turquoise	afeeee	hot pink	ff69b4
turquoise	40e0d0	pink	ffc0cb
cyan (*)	00ffff	light pink	ffb6c1
light cyan	e0ffff	pale violet red	db7093
dark cyan	008080	maroon (*)	b03060
cadet blue	5f9ea0	violet red	d02090
aquamarine	7fffd4	magenta (*)	ff00ff
seagreen	54ff9f	dark magenta	800080
light seagreen	20b2aa	violet	ee82ee
pale green	98fb98	plum	dda0dd
		orchid	da70d6
		blue violet	8a2be2
		purple	a020f0

Назву кольору in English, яка складається з кількох слів, слід писати без пробілу, *наприклад*, h1 { color: lightblue; }

Навчальне електронне видання

Денисенко Вікторія Юріївна

ВВЕДЕННЯ У Web UI РОЗРОБКУ

Навчальний посібник

Підписано до публікації 16.02.2018 р.
Гарнітура Times. Зам. №18-19
Видавець і виготовлювач:
Одеська державна академія будівництва та архітектури
Свідоцтво ДК № 4515 від 01.04.2013 р.
Україна, 65029, м. Одеса, вул. Дідріхсона, 4.
тел.: (048) 729-85-34, e-mail: rio@ogasa.org.ua