Вступ до Розробки Програмного Забезпечення

Яків Крамаренко

Зміст

Вступ

Частина I - Фронтенд

Вступ до Розробки Програмного Забезпечення

Яків Крамаренко

Ця книга - ранній доступ до курсу "Вступ до Розробки Програмного Забезпечення". Вона дає вступ до програмування на прикладах розробки веб-додатків використовуючи HTML, CSS та JavaScript. Книга має бути по силам будь-кому від дітей до їх батьків, з єдиною передумовою - бути впевненим користувачем комп'ютера. Вона має допомогти відчути на смак розробку невеликого але реального продукту і визначитись з бажаною роллю в IT - розробника, тестувальника, чи, можливо, когось ще;)

Книга доступна для безкоштовного завантаження а також добровільних пожертвувань за посиланням http://leanpub.com/intro-to-software-development.

Автор фото на обкладинці - Rafael Zamora.

Ця версія була опублікована 2018-08-07.

Передмова

У 2015 році я почав навчати на платних "офлайн" і онлайн IT-курсах (з програмування, автоматизації тестування, і т.д.). Я помітив кілька речей. По-перше, основна частина курсів на ринку, особливо безкоштовних, - були надто технічними та складними для студентів які починають свій шлях в IT з самого початку. По-друге, їм зазвичай складно визначитися який напрямок в IT обрати - менеджмент, бізнес-аналіз, дизайн, розробка, тестування, і т.д. У той час я почав подумувати про те, щоб створити курс який дасть введення в повний процес розробки програмного забезпечення і буде під силу майже для будь-кого від дітей до їхніх батьків, з єдиною передумовою - бути впевненим користувачем комп'ютера.

Ідея була в тому, щоб створити курс за допомогою якого студент зможе побудувати реальний веб-додаток з нуля. Де кожне заняття буде представляти один з етапів в повному циклі процесу розробки програмного забезпечення. Як визначає вікіпедія, Розробка Програмного Забезпечення -

> Це процес задумування, визначення, проектування, програмування, документування, тестування і виправлення помилок, пов'язаних зі створенням і підтримкою додатків, фреймворків або інших програмних компонентів. (Переведено з англійської)

Я почав роботу над цим курсом в 2016 році. Наступні заняття повинні були в нього увійти:

- Процес
- Бізнес-Аналіз
- Дизайн
- Розробка веб-клієнта (фронтенда)
- Розробка веб-сервера (Бекенд)
- Автоматизація тестування
- Тестування
- Розгортання Додатків ("Deployment")

Передбачалося, що студент познайомиться з кожним етапом процесу на прикладах створення реального веб-додатка з нуля - менеджера задач. Де кожне заняття покаже як планувати, аналізувати, проектувати, розробляти і тестувати основні функції програми, а за допомогою доступних вправ студент буде практикуватися в розширенні функціональності менеджера задач за допомогою доступних рад, частих питань і відповідей.

Згодом я зрозумів, що масштаб виконуваної роботи величезний. Особливо враховуючи мою зайнятість на інших проектах. До сих пір я закінчив тільки чернетку заняття "Процес" і заняття "Розробка веб-клієнта (Фронтенд)", без вправ. Швидше за все я опублікую чернетку заняття <Процес> в якості поста в блозі. А ця книга, принаймні на початку, стане домівкою для тих матеріалів курсу, які ближче до "програмування" (зміст може змінюватися):

- Розробка Веб-Клієнта - Фронтенд (HTML, CSS, JavaSript)
- Практики Забезпечення Якості. Автоматизація
- Розгортання Додатків (Deployment)
- Розробка Веб-Сервера - Бекенд
- Тестування

Частина книги про розробку веб-клієнта (фронтенд) вже доступна, поки що без вправ. Я планую тримати книгу завжди у вільному доступі і доступною для скачування. Але прогрес в розробці наступних занять і, нарешті, створення повного курсу, заснованого на книзі, буде залежати від пожертвувань. Чим більше я їх назбираю, тим менше часу мені потрібно буде витрачати на мої інші комерційні проекти, і віповідно буде більше часу і ресурсів для роботи над цими книгою і курсом. Пожертвування в будь-якому розмірі можна внести на сторінці книги (http://leanpub.com/intro-to-software-development) чи надсилаючи монети на мої гаманці:

Bitcoin - 1EyDGuW64YkJbZ8FW1yAkz6iLP8c6tCjn

Bitcoin Cash - qqp2g49z0eskfv5kyv53q4rf2huz8lqssqe47ysmyr

Ether - 0x7f2cAa79D1f1966d3CDd8295f8aF6028D66de00e

Програмування?

"Програмування" - це слово має всім нам бути знайоме. Всі ми користуємося побутовую технікою, яку в тій чи іншій мірі "програмуємо" на виконання потрібних дій у потрібному порядку. Наприклад, ми конструюємо програму прання для пральної машини: виставляємо час, інтенсивність прання, додаткове полоскання, і т.д. Це найпростіші форми алгоритмів, але все ж ми вже можемо називати себе "програмістами". Навіть наших дітей ми програмуємо свідомо чи не свідомо, коли виховуємо їх.

Комп'ютерні програмісти - роблять те ж саме тільки на набагато нижчому рівні.

Повернемось до пральної машини. Можливість налаштовувати режими прання ми отримали завдяки програмістам, які на етапі створення машини, запрограмували ці режими на більш детальному рівні. Ми не замислюємось над тим - скільки часу потрібно щоб нагріти воду - "внутрішня програма" сама ввімкне нагрівач і вимкне його при досягненні вказаної нами темпереатури, сама буде у потрібні моменти міняти воду, догрівати, і так далі...

Отже, якщо коротко, програмування – це процес написання програм, які описують передбачений хід подій у часі та порядок правил, що повинні виконуватись для проведення запланованого. Кінцевий набір таких "ходів і правил" також називають - алгоритмом.

Комп'ютерну ж програму можна визначити, як низку команд для комп'ютера, що становлять запис алгоритму однією з мов програмування.

Як ми використовуємо різні мови для спілкування з громадянами різних країн, різні підмножини мов чи то пак жаргони для спілкування на певні теми типу юрисприденції, медицини, економіки. Як використовуємо різні мови команд побутовій технці, мобільним пристроям, - так і програмісти використовують різні мови програмування та їх "підмножини" (бібліотеки) - для спілкування з комп'ютером з ціллю його "програмування" на виконання потрібних нам задач і алгоритмів.

В сучасному комп'ютерному програмуванні існує безліч напрямків у відповідності до типу розроблюваних програм:

- Розробка ігор - "gamedev"
- Розробка мобільних додатків
- настільних додатків
- Розробка "вбудованих систем" (embedded systems) - згадуємо пральну машину ;)
- Розробка веб-сайтів (веб-розробка)
- І т.д.

Саме на прикладі останньої "веб-розробки" - ми познайомимось з програмуванням в наступних заняттях.

Варто зауважити що термін додаток який ми вжили раніше - не зовсім точний і коректний, на відміну від "застосунок". Але оскільки термін "додаток" - більш популярний в народі, ми надалі в основному користуватимемось саме ним.

Веб-розробка?

Напевно, вам відомо про роботу зі стандартною програмою. Зазвичай, ви її завантажуєте з інтернету і встановлюєте на комп’ютер. Що ж таке веб-сайт? Це така спеціальна програма якій не потрібне встановлення — достатньо просто "відкрити" чи як кажуть "завантажити" її в Браузері, використовуючи її "адресу". Вона може бути одночасно відкрита багатьма користувачами з різних комп'ютерів і браузерів. Наприклад, сайтом Facebook одночасно можуть користуватися тисячі людей. Вона може видавати користувачам одну і ту ж чи різну інформацію в залежності від контексту, давати можливість користувачам взаємодіяти як з програмою, так і з іншими користувачами, що користуються нею. Веб-сайт може складатися з одної чи багатьох веб-сторінок, у кожної з яких - своя "адреса". Ця програма, яку різні користувачі можуть завантажувати в браузері - називається веб-клієнтом і, насправді, є тільки "частиною" веб-сайту. Але, існує і та частина "веб-сайту", яку ми не бачимо, але яка виступає його "мозком", який керує одночасною роботою всіх завантажених користувачами веб-клієнтів, надає їм потрібну інформацію, зберігає їх дані, і дає можливість їм взаємодіяти між собою. Цей "мозок" - називається веб-сервером.

Така "розумна" веб-програма, що складається з клієнту і серверу - ще відома під назвою "веб-додаток" чи "веб-застосунок" (англійською - "web application").

Відповідно, у розробці веб-додатків виділяють два під-напрямки:

- Фронтенд веб-разработка (клієнта) - "Frontend Web Development"
- Бекенд веб-разработка (сервера) - "Backend Web Development"

Напрямки бекенд і фронтенд – невід'ємні частини веб-розробки, і розробників які займаються і тим і іншим - іноді називають "фулстек" веб-розробниками ("full-stack web-developers"). Проте, через складність багатьох сучасних проектів і відповідних технологій — часто бекенд-розробники і фронтенд-розробники – це дві окремі гілки спеціалістів.

Іноді, також, в межах фронтенду виділяють окремо "верстку" веб-сторінок - частину роботи, пов'язану з розміткою їх структури у відповідності до макету дизайну сайту, заздалегідь намальованого за допомогою графічних редакторів.

Ми почнемо з простішого і "ближчого" до кінцевого користувача - фронтенду. На прикладі розробки веб-додатку для керування задачами - "таск-менеджера" ("task manager") — ми спочатку створимо невеличкий веб-клієнт, за допомогою якого користувач зможе створювати задачі, без можливості збереження задач між перезавантаженнями веб-сайту і без доступності створених задач з різних комп'ютерів. А після - ми візьмемось за розробку веб-серверу з ціллю зберігати задачі, щоб вони були доступні між перезавантаженнями веб-сайту, в тому числі і з різних комп'ютерів.

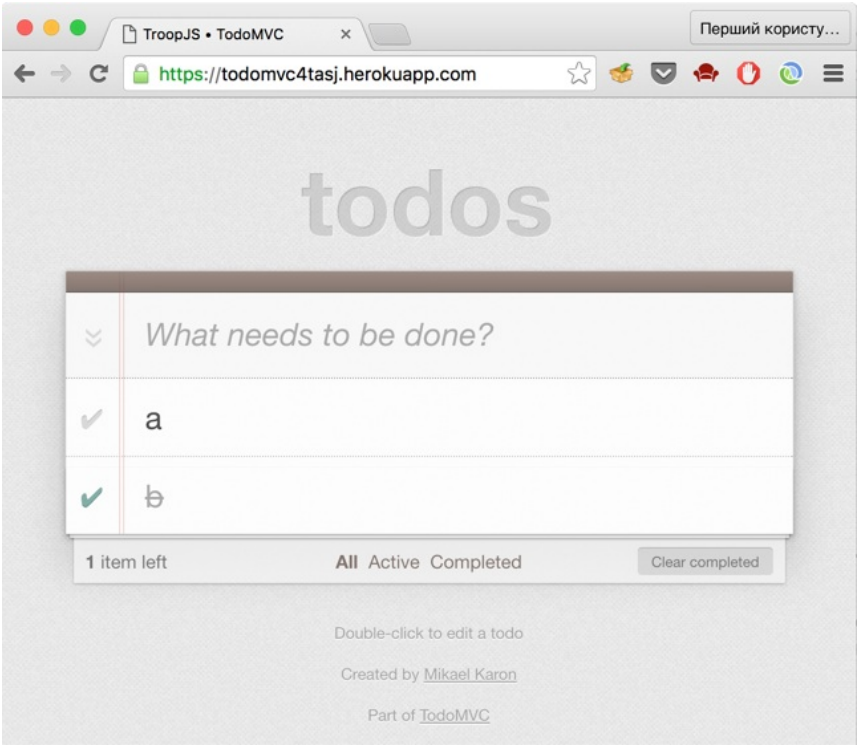
Варто також зазначити, що ми не будемо розглядати всі нюанси веб-розробки. Ми не будемо завжди слідувати точній термінології, і використовувати всі "останні фішки" розглянутих технологій. При цьому намагатимемось по можливості користуватись "останніми нововведеннями", якщо це спростить подачу матеріалу. Наша ціль - не вивчити базу програмування за такий обмежений час, а познайомитись з процесом роботи веб-розробника і відчути на практиці - наскільки це складно і цікаво.

HTML

Веб-сторінки - як їх бачимо ми, користувачі

Для того, щоб почати розробляти власні веб-додатки, нам потрібно на базовому рівні розуміти, як реалізовані "веб-сторінки".

Ми звикли під веб-сторінками розуміти те, що ми бачимо в браузері після завантаження веб-сайту:

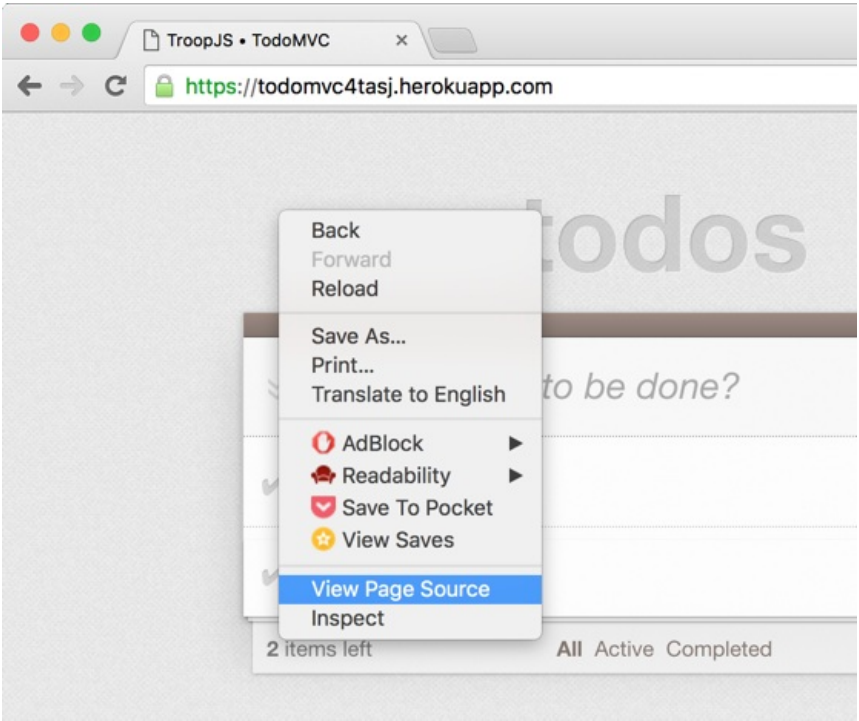


Веб-додаток TodoMVC

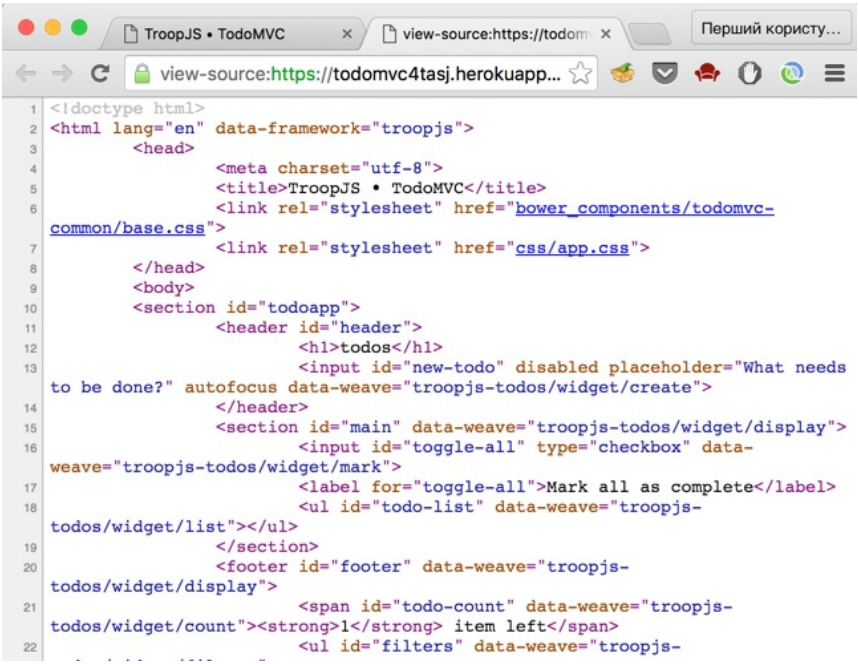
Але за всі ці гарненькі елементи - поля редагування, мітки з текстом, кнопки, чекбокси, посилання - ми маємо дякувати саме браузерам (Firefox, Chrome, Opera, Edge, Safari …), які вміють представляти внутрішню реалізацію сторінок в зрозумілому нам, людям, вигляді. Іншими словами - браузери - це перекладачі з "мови програмування веб-додатків" на "мову користувачів". Такий процес "перекладу" - також називають "інтерпретацією".

Веб-сторінки - як їх бачить браузер

А ось як виглядає веб-сторінка для браузеру:



Вибір "View page source" в контекстному меню сторінки



Вихідний код сторінки TodoMvc

Ось ця абракадабра - написана мовою розмітки гіпер-текстових документів (веб-сторінок) - HTML (Hyper Text Markup Language). Варто також зауважити, що в народі також можуть вживати відповідний англіцизм - "аштеемель". Це стосується більшості термінів такого типу. І ми надалі також для деяких англомовних термінів можемо вживати англіцизми, коли це буде зручно.

Слово "гіпер" означає, що ми маємо справу не просто з "лінійним" текстом, а з текстом, що може містити посилання (так звані "гіперпосилання") на інші ресурси - як в межах цієї ж сторінки, так і на інших сторінках за іншими адресами.

Для того, щоб зрозуміти секрет цієї мови, ми розглянемо більш простий приклад.

Тобі, мабуть, вже стало помітно, що веб-сторінка, яку ми завантажили раніше - пов’язана з задачами. Так і є - це вже повноцінно реалізований менеджер задач, що дозволяє створювати задачі, їх редагувати, видаляти, позначати "зробленими",

очищати "зроблені", фільтрувати. По суті, ми отримуємо повноцінну програму в браузері, а не просто веб-сторінку з інформацією без можливості її динамічно змінювати тут і зараз.

Одною з головних цілей цього посібнику - це попрактикуватись самотужки створювати подібні веб-додатки.

І зараз ми приступимо до реалізації основного функціоналу нашого менеджера задач - створення нових задач через введення їх тексту в текстове поле і натискання `Enter`, а також - відображення їх в списку нижче.

Ось як би міг виглядати базовий HTML-код нашого веб-додатку:

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input></input>
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```

Погляньте уважно на цю структуру. Її можна порівняти з великою шафою, що починається з `<html>` і закінчується на `</html>`. Як бачимо індентифікатори початку і кінця "шафи" - обрамлені спеціальними символиами - меньше (`<`) і більше (`>`). А щоб відрізнити "кінець" від "початку"б, у випадку "кінця" - замість `<` використовується `</`.

В середині шафи є дві основні секції - *head* і *body*, які теж відповідно мають свій початок (`<head>`, `<body>`) і кінець (`</head>`, `</body>`)

head - відіграє роль "паспорта" нашої веб-сторінки. В ньому поки що тільки одна "поличка" - з іменем нашої веб-сторінки - `<title>Todos</title>`. ("Todo" - є синонімом слова "задача" англійською…)

А в *body* - знаходиться тіло нашої веб-сторінки разом з усіма її "елементами-коробками".

Коробки можуть бути вкладені одна в одну.

В нашому випадку елементи списку

```
<li>watch lesson</li>
<li>do homework</li>
```

що відображають задачі, вкладені в свою окрему велику коробку - `<ul> … </ul>`, що відображає список задач.

`ul` доречі розшифровується як "**u**nordered **l**ist" — **н**евпорядкований **с**писок.

А `li` — як "**l**ist **i**tem" – **е**лемент **с**писку.

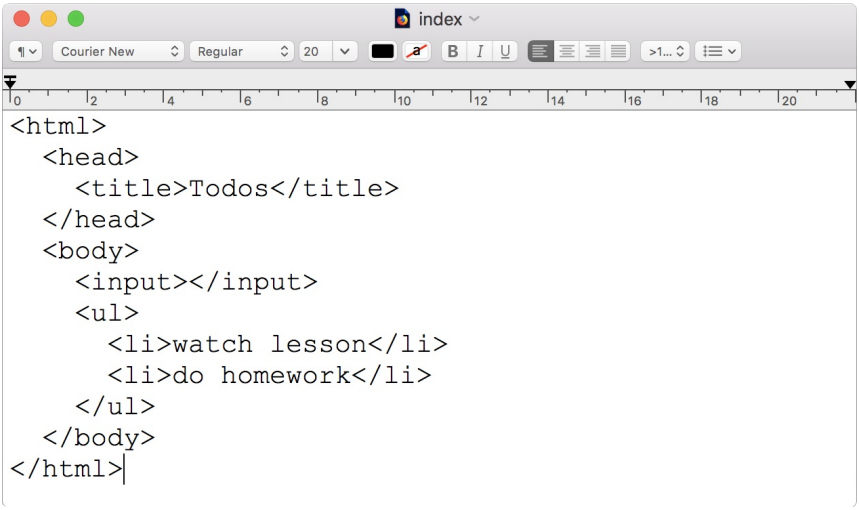
Як бачимо, "коробка" може мати один тільки текст - як наприклад елемент списку типу `<li>watch lesson</li>`.

А може і взагалі не мати, як з полем введення тексту: `<input></input>`. В таких випадках, коли в коробку "нічого не вкладено", можна скоротити запис до `<input/>`.

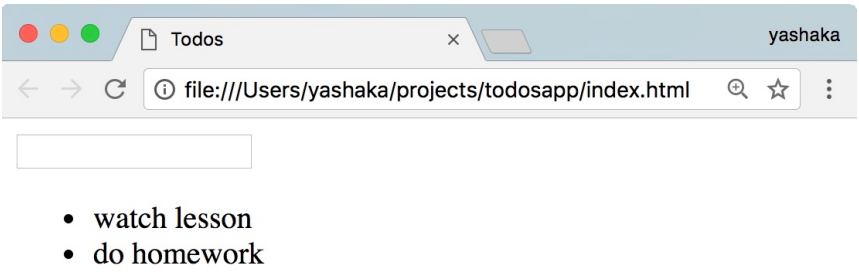
Всі ці наші коробки офіційно називаються **елементами**. Ідентифікатор їх початку називається **відкриваючим тегом** ("opening tag") — наприклад, `<li>`, а ідентифікатор кінця - **закриваючим тегом** ("closing tag") — наприклад, `</li>`.

Як бачиш, все, що дає HTML - це правила розкладання інформації про нашу веб-сторінку по вкладеним "коробкам" в такому от чудернацькому, але доволі "зручному" з точки зору комп'ютера чи браузера вигляді.

Давай тепер подивимось, як веб-сторінка з таким кодом виглядала б в браузері. Для цього збережемо код в файлі з розширенням `html` : `index.html` і відкриємо його в браузері:



Збереження html-прикладу в звичайному текстовому редакторі



Відкритий html-файл в браузері

Ось так.

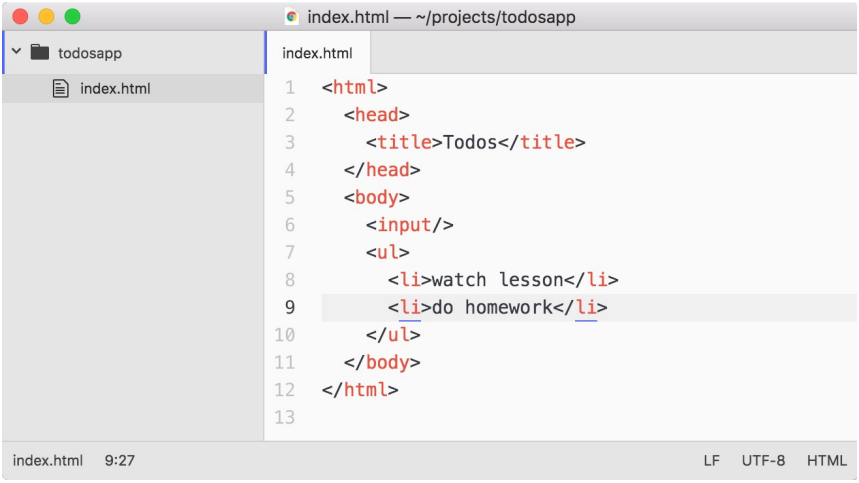
Мабуть ти думаєш - чому саме `index` ? Нехай це буде одним з твоїх домашніх завдань - розкрити цей секрет самотужки... Гугл в поміч ;)

Редактори Коду

Перед тим, як продовжити, давайте познайомимось з редакторами коду.

Будь-який код - це по суті текст, який можна редагувати в звичайному текстовому редакторі. Але в коду своя специфіка в порівнняні зі звичайним текстом. І як в текстових редакторах є додаткові функції типу підказок щодо граматичних помилок - так і в редакторах, спеціалізованих під редагування коду - є свої особливі функції, що допомагають працювати саме з кодом.

Ось як виглядає код index.html відкритий в одному з таких редакторів - Atom:

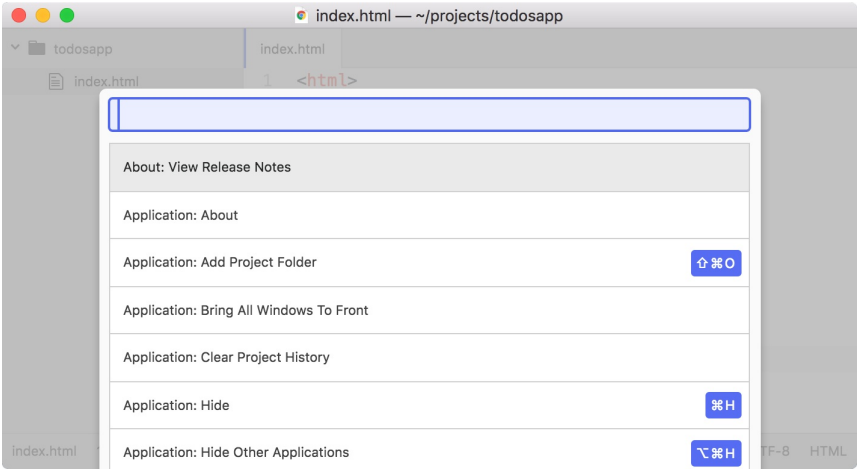


index.html в редакторі Atom

Як бачиш, одразу читати такий код стало легше через специфічну підсвітку синтаксису. Тепер в очах все не так розпливається через "просто текст" змішаний з "тегами елементів".

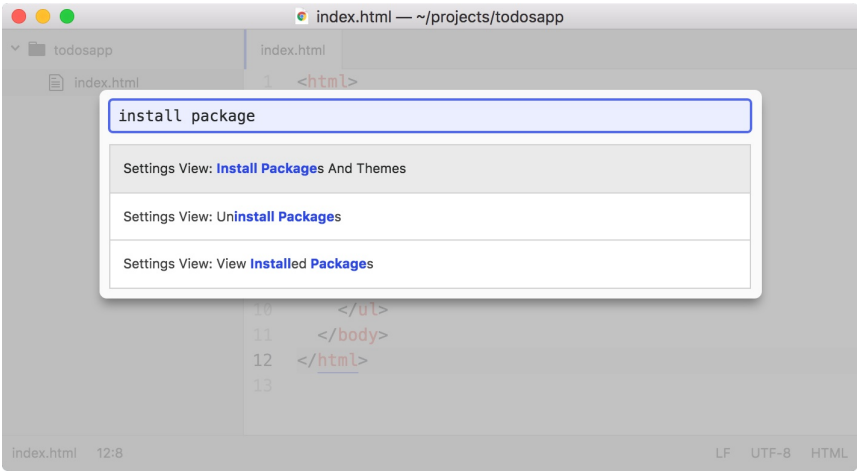
І ще дещо цікаве про "Атом". Існує безліч додаткових "модулів" чи так званих "пакетів" ("packages"), які полегшують "будні програміста", додаючи до базового функціоналу редактора додаткові функції. Наприклад, було б чудово мати додатковий неперервний "живий перегляд в браузері" нашого коду - можливість спостерігати в емуляторі браузера, як наш код буде інтерпретуватись після певних змін. Чи є такий пакет для "Atom"? — Не знаю, давай пошукаємо;)

Cmd-shit-p для Mac чи *ctrl-shift-p* для Windows відкриє діалог "Command Palette", в якому ми зможемо знайти потрібну нам фічу (від "feature" - певна частина функціоналу) редактора:



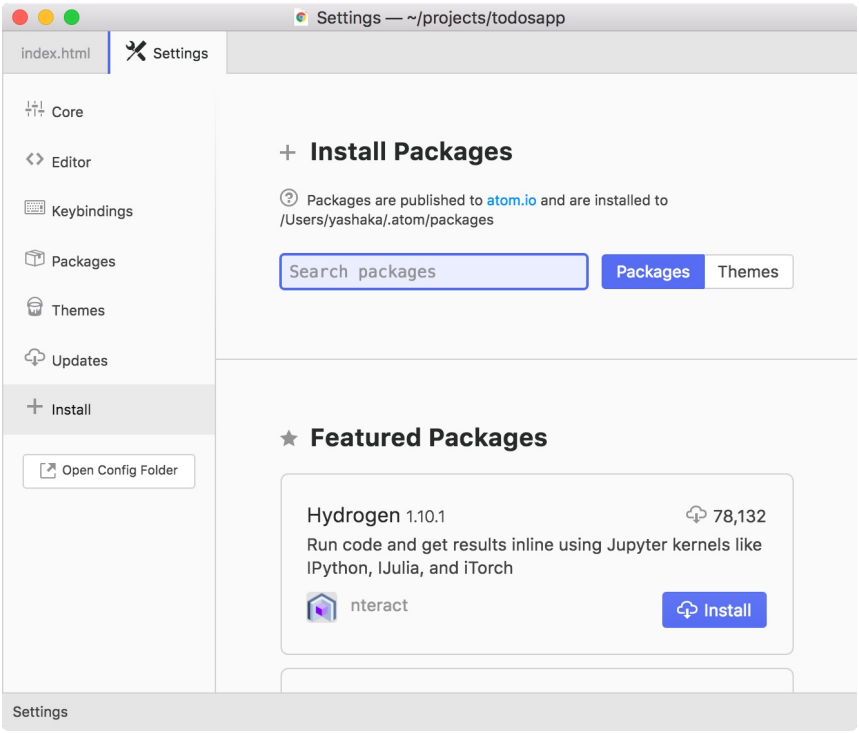
Command Palette

Давай пошукаємо "install package" ("встановити пакет"):



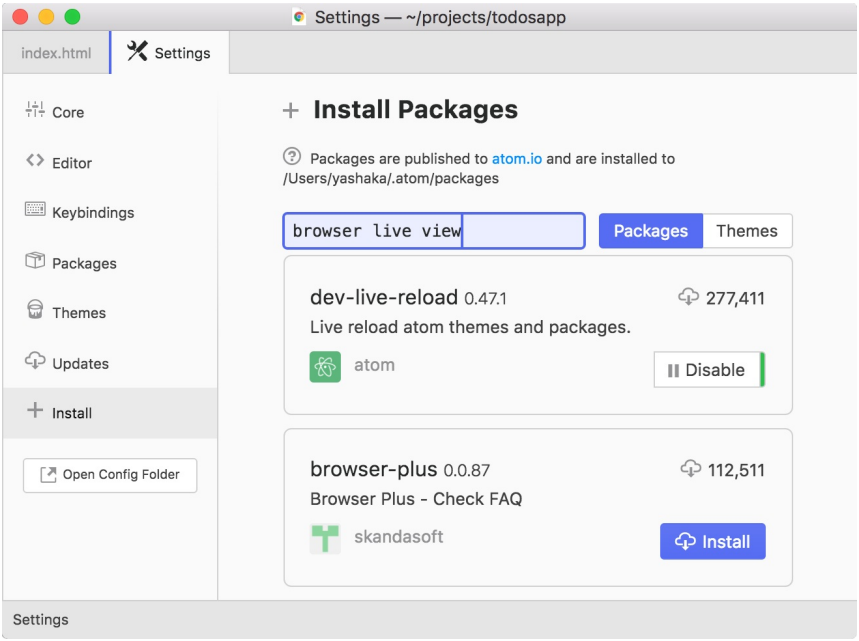
Пошук "install package"

Чудово, схоже що "Settings View: Install Packages And Themes" - це якраз те місце, де ми можемо перевірити, які пакети ми можемо встановити для "Atom":



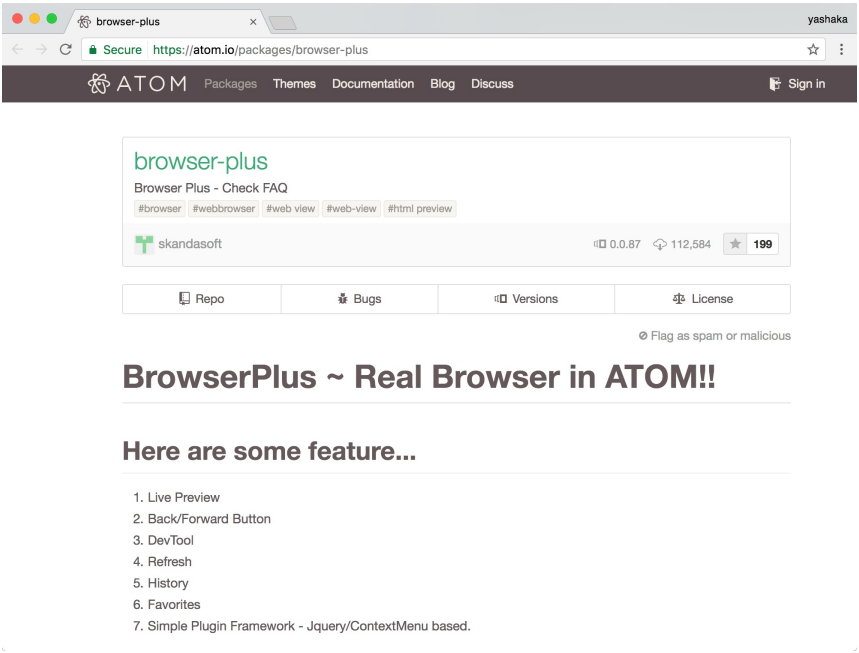
Settings>Install

Тепер перейдемо до "Search packages" і пошукаємо "browser live view" ("живий перегляд браузера"):



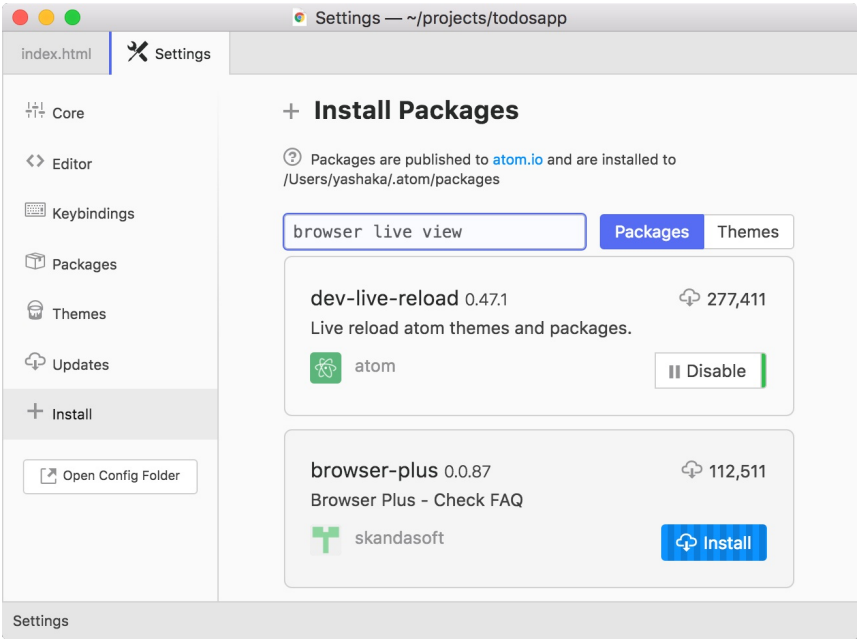
Пошук "browser live view"

О! Другий пункт в списку - "browser-plus". Якщо клікнути по його назві, ми отримаємо сторінку з детальною інформацією про пакет:

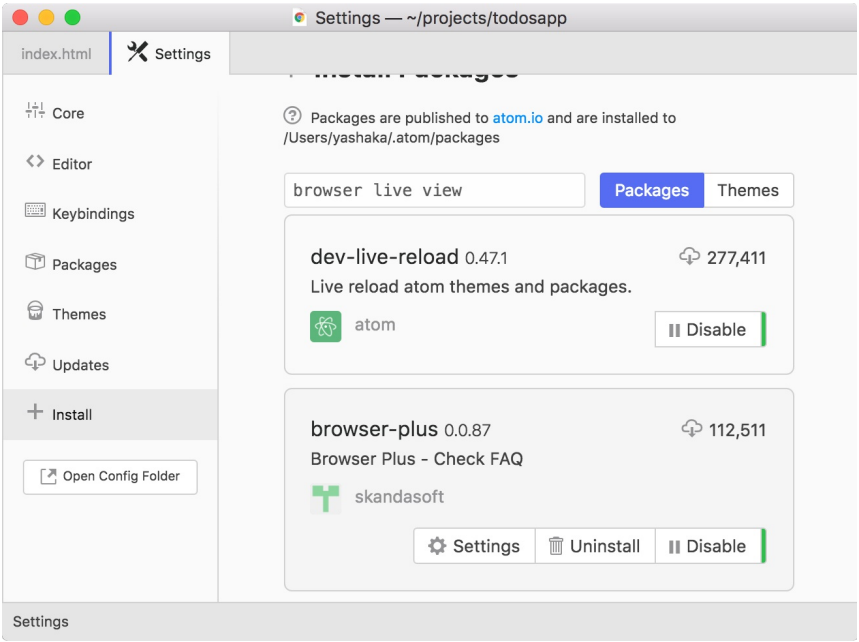


Інформація про browser-plus

"Real Browser in ATOM" ("Справжній Браузер в Атом")! І перша його особливість - "1. Live Preview". Думаю, це саме те що нам потрібно. Давай встановимо цей пакет.

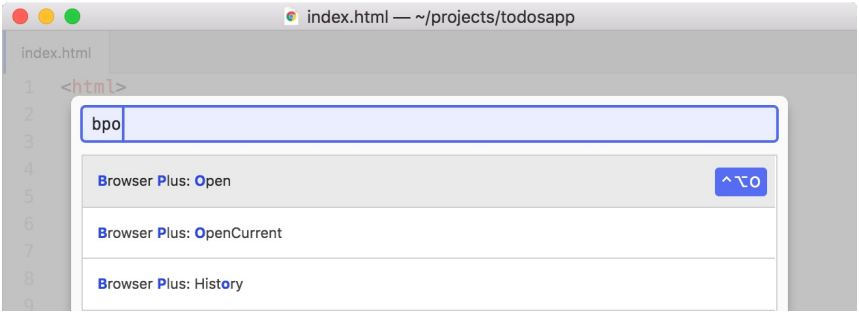


Встановлення пакета "browser-plus"

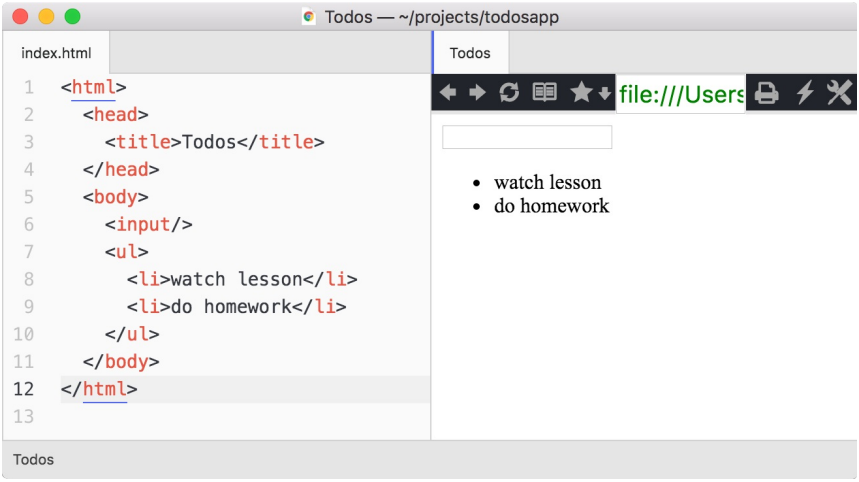


Встановлений пакет "browser-plus"

Тепер пакет "browser-plus" встановлено, і ми можемо включити його, використовуючи вже відому нам "Command Palette" (зверни увагу, що ми можемо шукати необхідну команду навіть по першим буквам слів з її назви):

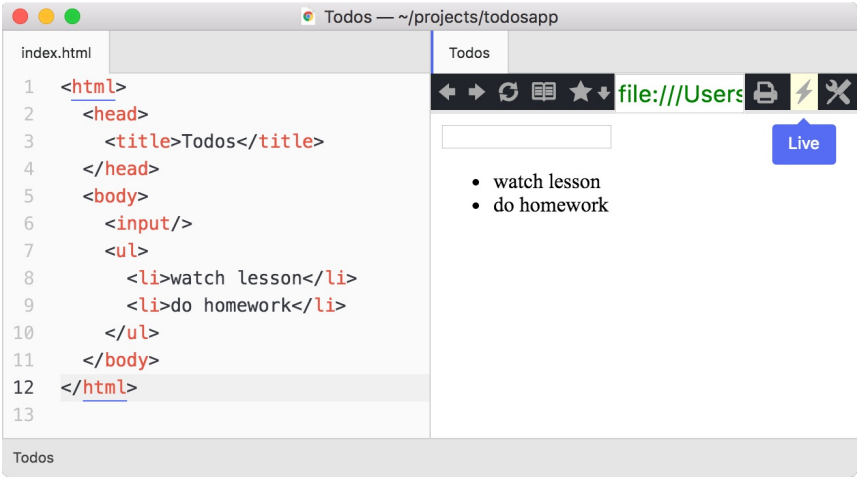


Активація "browser plus"



Активований "browser plus"

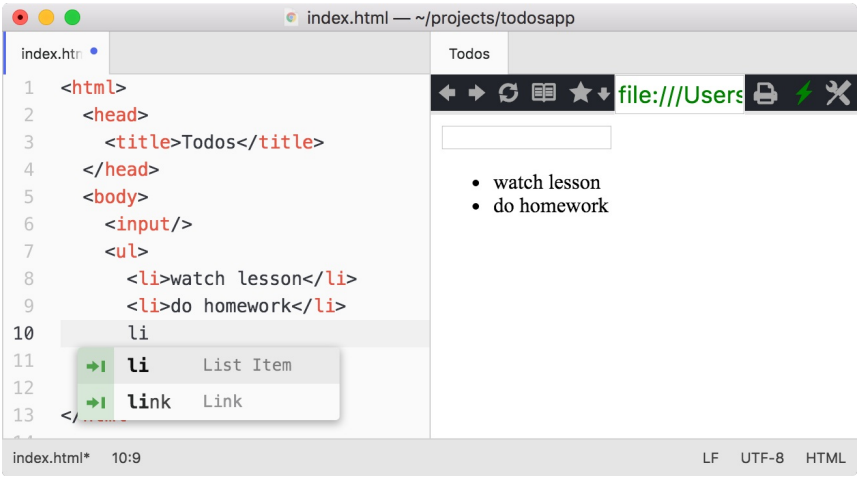
Щоб включити функцію "живого перегляду", можна нажати відповідну кнопку з зображенням блискавки:



Активація "live preview"

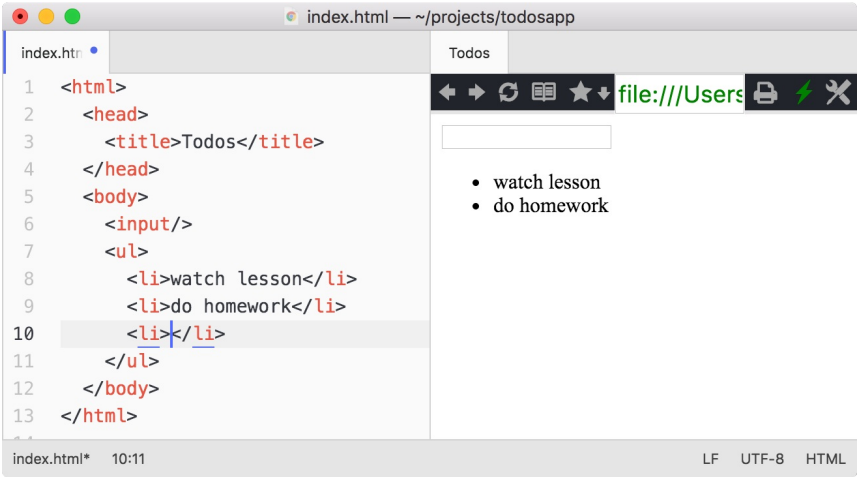
І тепер, якщо зберегти будь-яке нову зміну в файлі (через *cmd+s* на Mac чи *ctrl+s* у Windows), зміни одразу будуть відображені в емуляторі браузера.

Давай, наприклад, додамо ще одну задачу в список. Тут ми можемо познайомитись з ще одною особливістю "Atom" - автодоповненням. Давай почнемо додавати новий елемент `li`, просто вводячи символи `l` і `i` (без символів тегів - `<` і `>`):



Виклик підказок автодоповнення

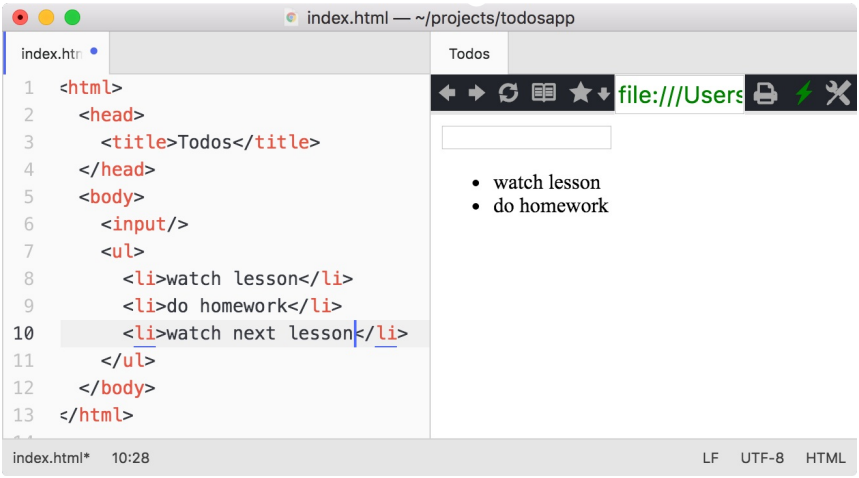
А тепер нажмемо `Tab` чи `Enter` , щоб завершити процес "автодоповнення":



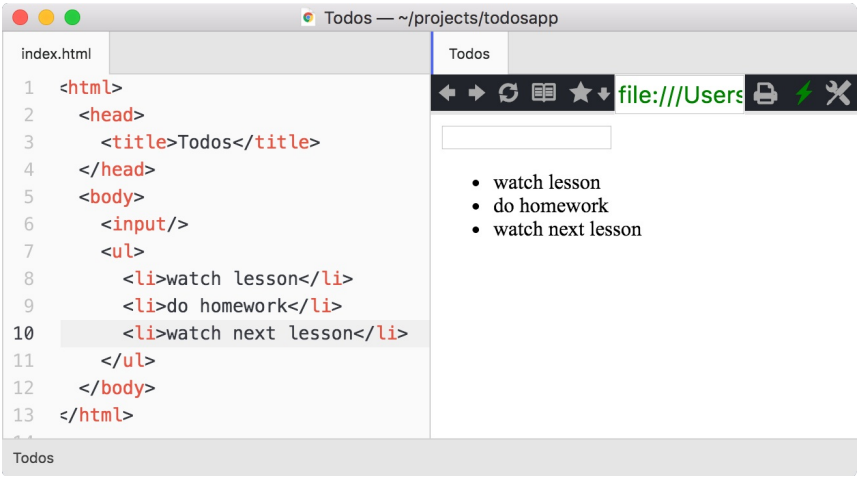
Завершення процесу автодоповнення

Редактор сам додасть символи відкриваючого тегу (‹ i ›), а також автоматично додасть закриваючий тег.

Тепер, додавши текст в нову задачу і нажав *cmd+s* на Mac чи *ctrl+s* у Windows (збереження змін), ми побачимо, що вони будуть відображені і в емуляторі браузера:



Зміни перед збереженням



Відображення змін після збереження

Круто, правда ж? :)

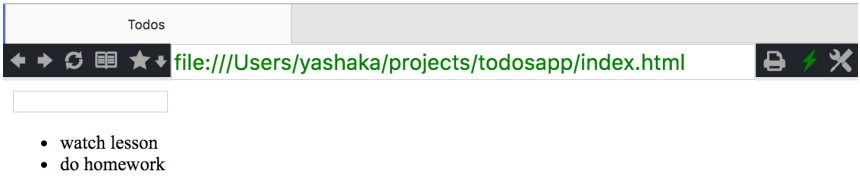
HTML Атрибути

Не для всіх знайомих нам елементів веб-сторінок існують свої специфічні теги.

Більшість елементів, що передбачають "фідбек від користувача" - у вигляді введеного тексту, вибраного чекбоксу чи радіо-батону, натиснутої кнопки - всі вони реалізовуються через елемент з тегом `input` .

Раніше ми вже використали елемент `input` для представлення поля редагування, в яке користувач вводитиме текст для нової задачі.

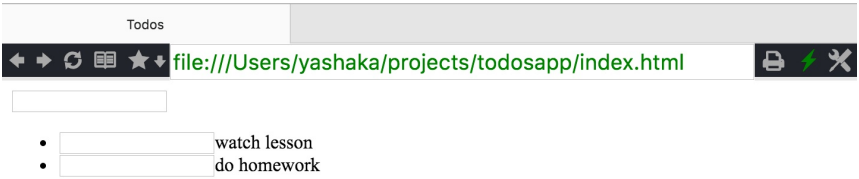
```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input />
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```



Елемент `input` як текстове поле

Давай тепер, для прикладу (хоч це і не буде нам потрібно в контексті розробки саме нашої версії таск-менеджера) - додамо чекбокси для кожної з задач, щоб дати можливість їх "завершувати", тобто відмічати як "зроблені":

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input />
    <ul>
      <li><input />watch lesson</li>
      <li><input />do homework</li>
    </ul>
  </body>
</html>
```



Елементи `input` для завершения задач

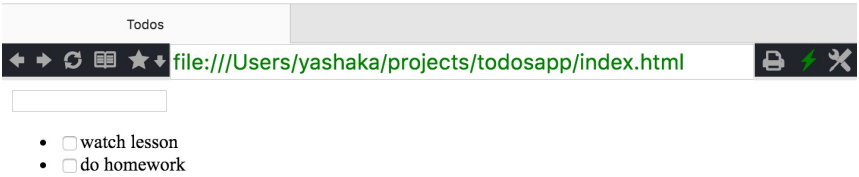
Але як тепер вказати, що новододані елементи `input` для задач - саме чекбокси?

В таких та інших ситуаціях, коли потрібно надати додаткову інформацію щодо елемента, чи то пак додати елементам своєї "індивідуальності", використовуються **html-атрибути**.

Атрибути бувають "загальними", які можна додати до будь-якого елемента, а бувають "специфічні тільки певним елементам", тобто має сенс їх додавати тільки до елементів з певними тегами. Останнє стосується наприклад атрибута `type` актуального саме

для елемента `input`, який якраз і дозволить нам вказати, що наш елемент `input` - саме чекбокс:

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input />
    <ul>
      <li><input type="checkbox" />watch lesson</li>
      <li><input type="checkbox" />do homework</li>
    </ul>
  </body>
</html>
```

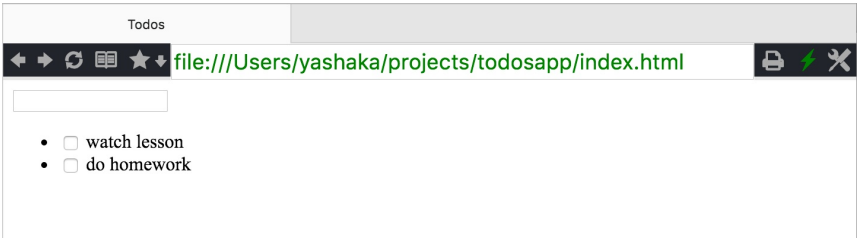


Елементи `input` як чекбокси

Тільки для кращої структурності, аби більш точно виділити кожну частину функціональності задачі - помістимо текст наших задач в середину своїх власних елементів, що грають роль "етикеток", "міток" чи "написів" для задач - елементи `label` :

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input/>
    <ul>
```

```
      <li>
        <input type="checkbox" />
        <label>watch lesson</label>
      </li>
      <li>
        <input type="checkbox" />
        <label>do homework</label>
      </li>
    </ul>
  </body>
</html>
```



Елементи label

Тепер функціональність "завершення задачі" і функціональність "представлення тексту задачі" - відображені в розмітці окремими елементами. Це в майбутньому дозволить більш зручно і точно знаходити "потрібну функціональність" в коді. Ідея така ж як і "розкласти по поличкам в шафі розкидані речі";)

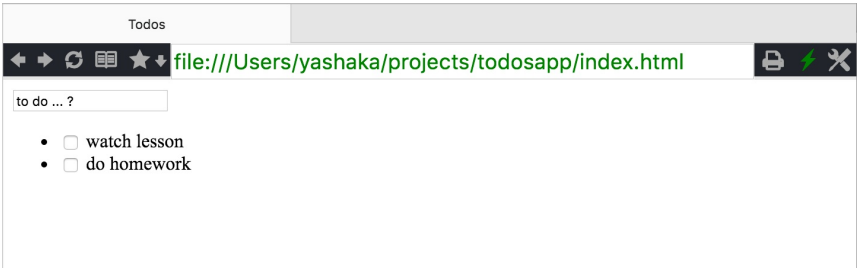
Давай познайомимося ще з деякими атрибутами такого "функціонального" типу, які додають нашим елементам важливих "функцій".

Як тобі ідея вписати в поле редагування якийсь текст для підказки користувачу?

Ось як це можна зробити за допомогою атрибута `value` для елемента `input`:

```
<html>
  <head>
    <title>Todos</title>
  </head>
```

```
  <body>
    <input value="to do ... ?" />
    <ul>
      <li>
        <input type="checkbox" />
        <label>watch lesson</label>
      </li>
      <li>
        <input type="checkbox" />
        <label>do homework</label>
      </li>
    </ul>
  </body>
</html>
```



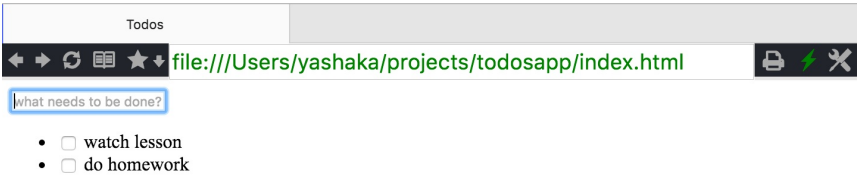
Елемент `input` з атрибутом `value`

Працює і "обернений зв'язок" - якщо ми введемо текст в поле редагування в браузері, то цей текст "буде збережено" і в значенні атрибута `value` . Хоча, поки що ми не знаємо способу, як це перевірити. Але ми повернемось до цього трошки пізніше, коли скористаємось цією особливістю, щоб "піддивитись" значення атрибуту `value` текстового поля після натискання користувачем `Enter` , і потім додамо нову задачу з підглянутим текстом у список ;)

Хвилинку, але ж вписавши в поле текст, ми тепер змушуємо користувача кожен раз його видаляти перед тим як ввести свій…

Є простий спосіб це виправити. Виявляється, є інший атрибут - `placeholder` - який виконує саме роль "підказки користувачу", яка не заважає введенню нового тексту:

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input placeholder="what needs to be done?" />
    <ul>
      <li>
        <input type="checkbox" />
        <label>watch lesson</label>
      </li>
      <li>
        <input type="checkbox" />
        <label>do homework</label>
      </li>
    </ul>
  </body>
</html>
```



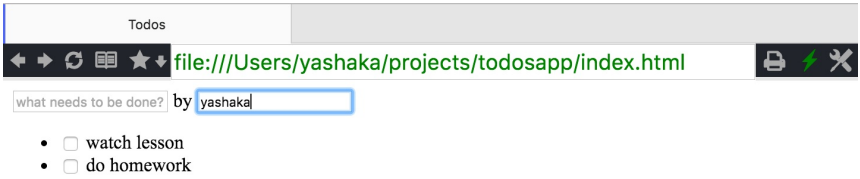
Елемент `input` з атрибутом `placeholder`

Отож, за допомогою атрибутів ми можемо надати Браузеру додаткову інформацію про елемент. Браузер вміє інтерпретувати специфічний набір атрибутів елемента і їх значень, і відповідно змінювати відображення елемента на сторінці. Наприклад, якщо у елемента `input` є атрибут `type="checkbox"` , то Браузер відобразить його як

чекбокс. Аде не для всіх подібних наших побажань про зміну стиля елемента чи певної поведінки, пов'язаної з ним, - існують заготовлені атрибути і їх значення. От наприклад, нам би не завадило змінити стиль елемента `input` для вводу тексту нової задачі так, щоб він відображався по центру, а головне - навчити його реагувати на натиск `Enter` створюючи нову задачу з введеним раніше текстом. Для цього прийдеться написати окремий код в окремих файлах на інших мовах, який сам буде знаходити потрібні елементи і змінювати їх стиль і поведінку. Ми займемось цим трохи пізніше, а зараз давай подумаємо от над чим. Як такий код зможе швидко знайти саме наш елемент `input` серед інших елементів `input` ? Як відрізнити елемент `input` як "текстове поле" від елемента `input` який чекбокс?

Можна було б навчити цей код шукати *"елемент input який не є чекбоксом"*, але що якщо у нас буде ще одне поле введення тексту яке, наприклад, відображатиме ім’я користувача за яким закріплена задача?:

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input placeholder="what needs to be done?" />
    by
    <input />
    <ul>
      <li>
        <input type="checkbox" />
        <label>watch lesson</label>
      </li>
      <li>
        <input type="checkbox" />
        <label>do homework</label>
      </li>
    </ul>
  </body>
</html>
```



Два текстових поля

Як тепер відрізнити два поля введення? Непоганою ідеєю буде використати пошук в стилі *"знайти елемент input в якого є атрибут placeholder - 'What needs to be done'"*

Але що якщо наша веб-сторінка підтримує 10 мов? Не заскладно буде перераховувати:

> *"знайти елемент input в якого є атрибут placeholder зі значенням 'What needs to be done?' або 'O que precisa ser feito?' або 'Що потрібно зробити?', або '需要做什麼呢' або ..."*

І варто врахувати, що з часом наша веб-сторінка може ускладнюватись, і додаватимуться нові елементи `input`, які ще більше ускладнюватимуть задачу пошуку потрібного елементу нашим додатковим кодом для "додавання задачі по натисканню `Enter`"

Підсумовуючи сказане - оскільки ми можемо мати елементи одного типу (тобто з одним і тим же тегом), але для різних цілей, нам потрібен спосіб однозначного маркування таких елементів щоб їх розрізняти.

Для цього існують спеціальні атрибути:

- Атрибут `id`, що надається унікальним елементам в межах веб-сторінки
- Атрибут `class`, що надається елементам що належать певній групі

І саме завдяки таким атрибутам ми можемо зв'язати необхідний функціонал з відповідними елементами в межах додаткового кода, згаданого раніше, який нам ще доведеться реалізувати.

Промаркуємо і наші елементи `input` відповідно до їх ролей:

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input id="new-todo" placeholder="what needs to be done?" />
    by
    <input id="assignee" />
    <ul id="todo-list">
      <li>
        <input class="toggle" type="checkbox" />
        <label>watch lesson</label>
      </li>
      <li>
        <input class="toggle" type="checkbox" />
        <label>do homework</label>
      </li>
    </ul>
  </body>
</html>
```

Тепер у нас завжди є однозначний спосіб відрізнити одне поле введення тексту від іншого. І навіть зробити "вибірку" групи чекбоксів, що відповідають классу `"toggle"` . Це може бути корисно для надання їм специфічного стилю.

Саме налагодженням стилю нашого веб-додатку ми і займемось в наступному розділі;)

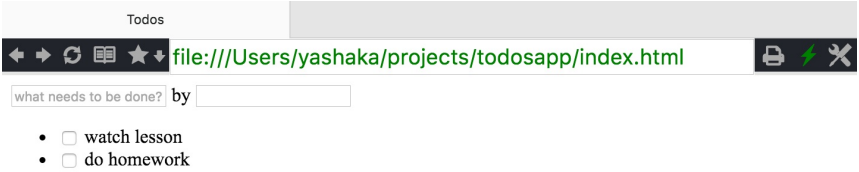
Додаємо унікальний стиль (CSS)

Додавання спеціальних стилей до html-сторінки

HTML дав нам змогу структурувати зміст сторінки - розмітити її дані, відобразивши їх ієрархічну вкладену структуру.

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input id="new-todo" placeholder="what needs to be done?" />
    by
    <input id="assignee" />
    <ul id="todo-list">
      <li>
        <input class="toggle" type="checkbox" />
        <label>watch lesson</label>
      </li>
      <li>
        <input class="toggle" type="checkbox" />
        <label>do homework</label>
      </li>
    </ul>
  </body>
</html>
```

Браузер вміє візуально представити такі "структуровані дані" відповідно до "стилю за замовчуванням".



Стиль за замовчуванням

Звісно, було б чудово налаштовувати стиль візуального представлення даних веб-сторінок до нашого смаку.

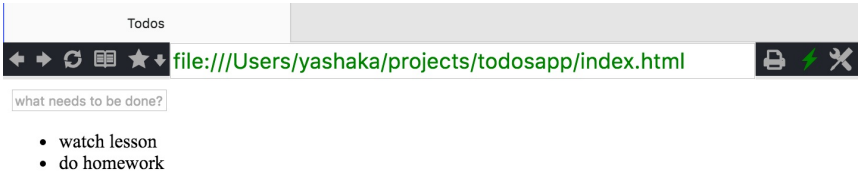
Це можна зробити за допомогою іншого інструменту веб-розробки - **Каскадних Таблиць Стилів** (Cascading Style Sheets) чи коротко **CSS**, або використовуючи популярний англіцизм - "ЦСС").

Це спеціальна мова, що дозволяє у вигляді набору правил описувати стилістичні властивості відповідних елементів.

Зазвичай стилі описують в окремих файлах з розширеням `.css` .

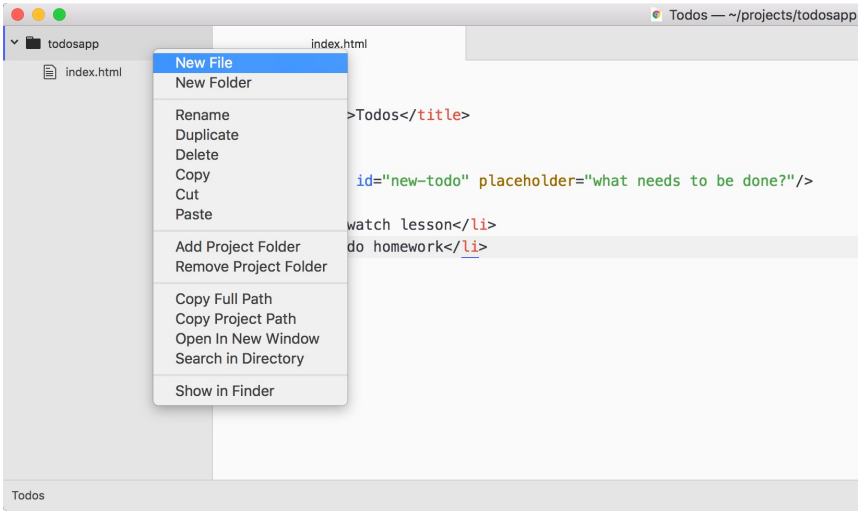
Перед тим, як написати стилі для нашого веб-додатку, ми спростимо наш html-код — до тої структури, яка необхідна для реалізації функціоналу тільки "додавання задач". Це нам також і полегшить розуміння всього процесу.

```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input id="new-todo" placeholder="what needs to be done?" />
    <ul id="todo-list">
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```

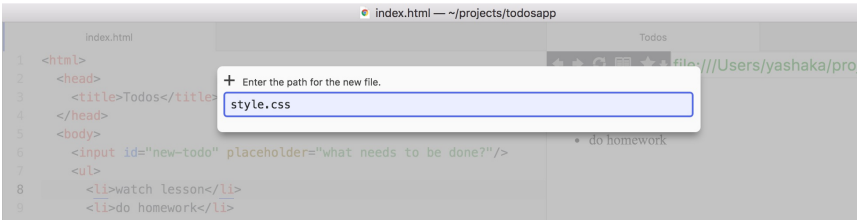


Спрощений код у браузері

Тепер в рамках нашого проекту (ми можемо включати/виключати панель з "деревом" проекту, так званим "tree view", — за допомогою "cmnd+\" для Mac чи "ctrl+\" для Windows) створимо новий файл `style.css` :



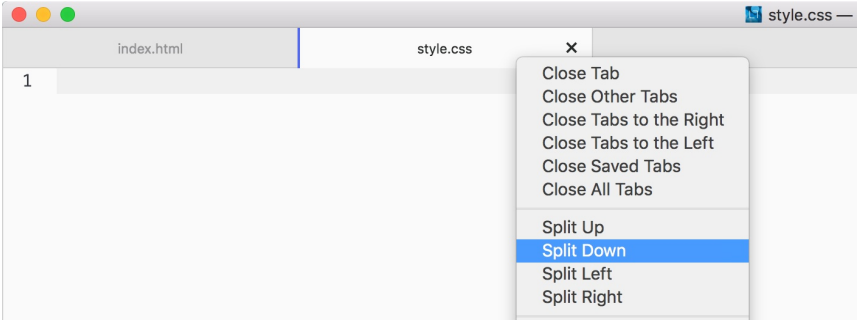
Створення нового файлу через контекстне меню з панелі дерева проекту



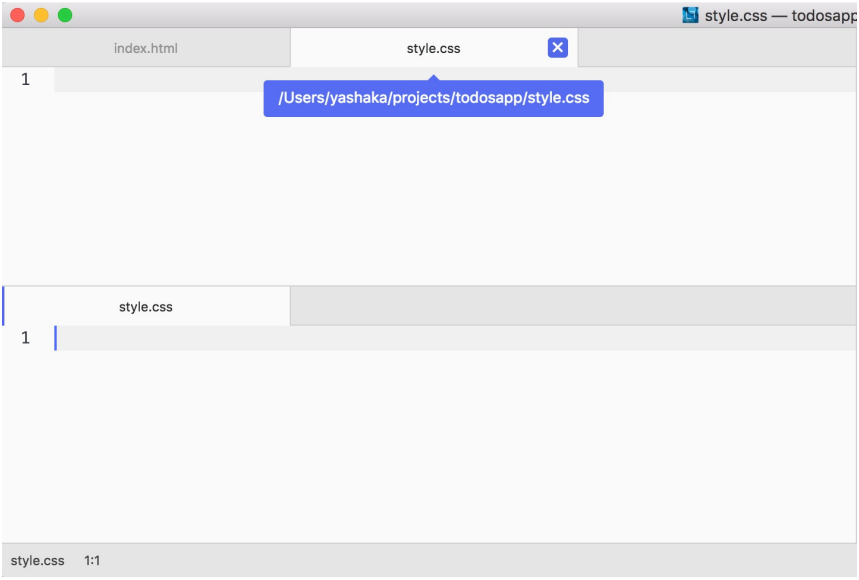
Задання імені нового файлу

Створений новий файл

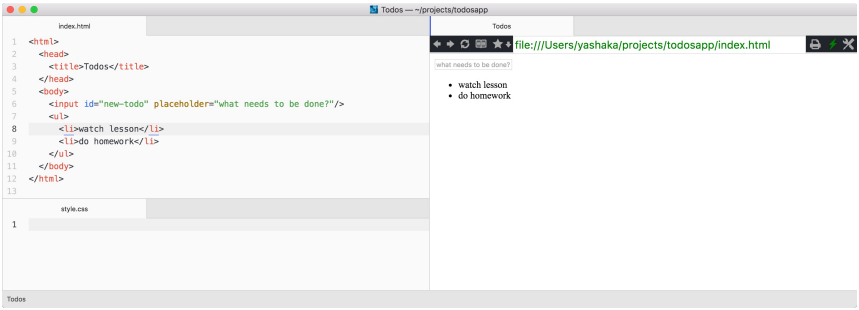
Для зруності давай закриємо ліву панель з деревом проекту ("cmnd+\" для Mac чи "ctrl+\" для Windows) и пересунемо нову відкриту вкладку в окрмеу секцію в редакторі коду за допомогою елемента контекстного меню "splitting the window down":



закриємо "дублікат":



і, нарешті, підправимо розмір вкладок у відповідності з нашим смаком:



Ми ще не написали жодного "правила застосування стилю", але давай одразу підключимо наш файл з стилями "css" до html сторінки, щоб вбудований в редактор браузер міг одразу їх "застосовувати" по ходу написання коду.

"Підключання" полягає в додаванні спеціального "реєстрового запису"

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

в "паспорт нашої сторінки" - секцію `head` :

```
<html>
  <head>
    <title>Todos</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <input id="new-todo" placeholder="what needs to be done?" />
    <ul id="todo-list">
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```

Правила CSS

Ну що ж, почнемо "прикрашати" нашу веб-сторінку:)

Почнемо з поля введення тексту - вам не здається, що воно занадто коротке?

Давайте "розтягнемо" його на всю ширину сторінки.

Сформуємо спочатку **правило стилю** звичайною українською мовою:

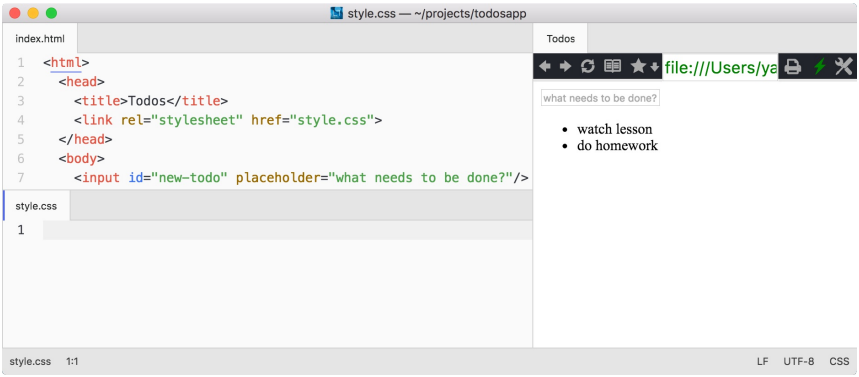
> Елемент з `id="new-todo"` повинен мати ширину у весь доступний простір

Чи більш точною "технічною" мовою:

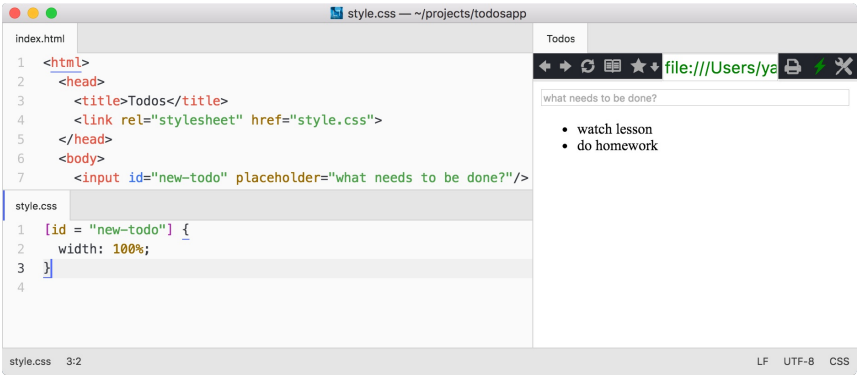
> Елемент з `id="new-todo"` повинен мати ширину в 100% доступного простору

А ось і переклад на "мову CSS"

```
[id = "new-todo"] {
  width: 100%;
}
```



Елемент input з шириною за замовченням



Елемент input з шириною в 100%

Як бачимо, переклад доволі однозначний:

> Елемент з id="new-todo" ...

```
[id = "new-todo"] {
  width: 100%;
}
```

... повинен мати ...

```
[id = "new-todo"] {
  width: 100%;
}
```

... ширину в 100% доступного простору

```
[id = "new-todo"] {
  width: 100%;
}
```

CSS-селектори

Правило починається з **селектора**, який визначає вибірку елементів до яких правило буде застосоване:

```
[id="new-todo"] {
  width: 100%;
}
```

В даному випадку - до елементів у яких є атрибут `id="new-todo"` (у нас таких - рівно один, для цього ми раніше і використали атрибут для унікальної ідентифікації елемента - `id`).

Синтаксис квадратних дужок

```
[id="new-todo"] {
  width: 100%;
}
```

є універсальним способом сказати: *"елемент чи елементи у яких атрибут має таке то значення"*. Наприклад, ми могли б "знайти наш елемент" і через **вибірку за двома атрибутами**:

```
[id="new-todo"][placeholder="What needs to be done?"] {
  width: 100%;
```

```
}
```

Також селектор дозволяє відрізнити "пошук по атрибутам" від **"пошуку по тегу елемента"**:

```
input[id="new-todo"][placeholder="What needs to be done?"] {
  width: 100%;
}
```

Тепер наш селектор каже:

> *"знайти елемент(и)* ***з тегом input****, з атрибутом* `id="new-todo"` *, і атрибутом* `placeholder="What needs to be done?"`

В будь-якому випадку, оскільки `id` - спеціальний атрибут який має бути унікальним для елемента на сторінки - то ми можемо обмежитись пошуком лише по одному атрибуту - `id` :

```
[id="new-todo"] {
  width: 100%;
}
```

Більше того, виявляється, так часто шукають елементи саме по атрибуту id, що в CSS передбачили скорочений синтаксис:

замість

```
[id="new-todo"] {
  width: 100%;
}
```

можна писати просто

```
#new-todo {
  width: 100%;
}
```

Властивості CSS

Продовжимо розбирати синтаксис CSS-правил далі…

Елемент з `id="new-todo"` повинен мати ширину в 100% вільного простору

```
#new-todo {
  width: 100%;
}
```

За селектором - в фігурних дужках - слідує **блок визначення правила**:

```
#new-todo {
     width: 100%;
}
```

в якому перераховані "стилістичні" властивості, які потрібно "встановити" для знайдених елементів чи елемента:

```
#new-todo {
     width: 100%;
}
```

Властивості в списку визначаються у відповідності до наступного синтаксису:

ім'я властивості

```
#new-todo {
     width: 100%;
}
```

двокрапка

```
#new-todo {
     width: 100%;
}
```

значення

```
#new-todo {
     width: 100%;
}
```

крапка з комою

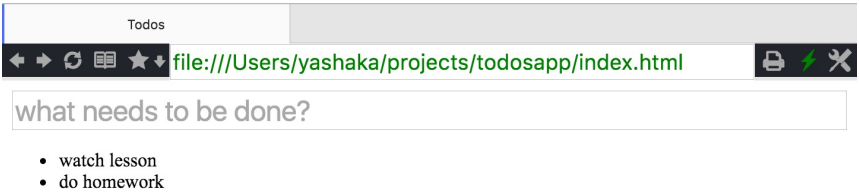
```
#new-todo {
     width: 100%;
}
```

Більше прикладів. Налаштовуємо стиль шрифту

Давай додамо в список ще трохи властивостей... :)

Збільшимо розмір шрифту - до 24 пікселів:

```
#new-todo {
  width: 100%;
  font-size: 24px;
}
```



Зробимо шрифт курсивом:

```
#new-todo {
  width: 100%;
  font-size: 24px;
  font-style: italic;
}
```