Введение в Разработку Программного Обеспечения

Яков Крамаренко

Содержание

Вступление

Часть I - Фронтенд

Введение в Разработку ПО

Яков Крамаренко

Эта книга - ранний доступ к курсу "Введение в Разработку Программного Обеспечения". Она дает введение в программирование на примерах разработки веб-приложений используя HTML, CSS и JavaScript. Книга должна быть по силам любому от детей до их родителей, с единственным предусловием - быть уверенным пользователем компьютера. Она должна помочь попробовать на вкус разработку небольшого но реального продукта и определиться с желаемой ролью в ИТ - разработчика, тестировщика, либо, возможно, кого то еще;)

Книга доступна для бесплатного скачивания а также добровольных пожертвований по ссылке http://leanpub.com/intro-to-software-development.

Автор фото на обложке - Rafael Zamora.

Эта версия была опубликована 2018-08-07.

Предисловие

В 2015 году я начал обучать на платных "офлайн" и онлайн ИТ-курсах (по программированию, автоматизации тестирования, и т.д.). Я заметил несколько вещей. Во-первых, основная часть курсов на рынке, особенно бесплатных, — были чересчур техническими и сложными для студентов которые начинают свой путь в ИТ с самого начала. Во-вторых, им обычно сложно определиться какое направление в ИТ избрать - менеджмент, бизнес-анализ, дизайн, разработка, тестирование, и т.д. В то время я начал подумывать о том, чтобы создать курс который даст введение в полный процесс разработки программного обеспечения и будет по силам почти для любого от детей до их родителей, с единственным предусловием - быть уверенным пользователем компьютера.

Идея была в том, чтобы создать курс с помощью которого студент сможет построить реальное приложение с нуля. Где каждый урок будет представлять один из этапов в полном цикле процесса разработки программного обеспечения. Как определяет википедия, Разработка Програмного Обеспечения —

> это процесс задумывания, определения, проектирования, программирования, документирования, тестирования и исправления ошибок, связанных с созданием и поддержкой приложений, фреймворков или других программных компонентов. (Переведено с английского)

Я начал работу над этим курсом в 2016 году. Следующие занятия должны были в него войти:

- Процесс
- Бизнес-Анализ
- Дизайн
- Разработка веб-клиента (Фронтенд)
- Разработка веб-сервера (Бекенд)
- Автоматизация Тестирования
- Тестирование
- Развертывание Приложения ("Deployment")

Предполагалось, что студент познакомится с каждым этапом процесса на примерах создания реального веб-приложения с нуля - менеджера задач. Где каждый урок покажет как планировать, анализировать, проектировать, разрабатывать и тестировать основные функции приложения, а с помощью доступных упражнений студент будет практиковаться в расширении функциональности менеджера задач с помощью доступных советов, частых вопросов и ответов.

Со временем я понял, что масштаб выполняемой работы огромен. Особенно учитывая мою загрузку на других проектах. До сих пор я закончил только черновик урока "Процесс" и урок "Разработка веб-клиента (Фронтенд)", без упражнений. Скорее всего я опубликую черновик урока <Процесс> в качестве поста в блоге. А эта книга, по крайней мере в начале, станет домом для тех материалов курса, которые ближе к "программированию" (содержание может меняться):

- Разработка Веб-Клиента - Фронтенд (HTML, CSS, JavaSript)
- Практики Обеспечения Качества. Автоматизация
- Развертывание Приложения (Deployment)
- Разработка Веб-Сервера - Бекенд
- Тестирование

Часть книги о разработке веб-клиента (Фронтенд) уже доступна (без упражнений). Я планирую держать книгу всегда в свободном доступе и доступной для скачивания. Но прогресс в разработке следующих уроков и, наконец, создание полного курса, основанного на книге, будет зависеть от пожертвований. Чем больше я их собираю, тем меньше времени мне нужно будет тратить на мои другие коммерческие проекты, и поэтому у меня будет больше времени для работы над книгой и курсом. Пожертвования в любом размере можно внести на странице книги (http://leanpub.com/intro-to-software-development) или перечислив монет на мои кошельки:

Bitcoin - 1EyDGuW64YkJbZ8FW1yAkz6iLP8c6tCjn

Bitcoin Cash - qqp2g49z0eskfv5kyv53q4rf2huz8lqssqe47ysmyr

Ether - 0x7f2cAa79D1f1966d3CDd8295f8aF6028D66de00e

Программирование?

"Программирование" - слово, которое должно быть нам всем знакомо. Мы все используем бытовую технику, которую программируем для выполнения необходимых действий в нужном порядке. Например, мы создаем программу стирки для стиральной машины: задаем время, интенсивность, дополнительное полоскание и т.д. Такие действия являются простейшими формами алгоритмов, и мы уже можем гордо назвать себя "программистами". Даже наших детей мы "программируем", когда воспитываем их.

Компьютерные программисты делают то же самое, но на гораздо более низком уровне.

Вернемся к нашей стиральной машине. Кто же создал эти настраиваемые режимы стирки? - Это сделали программисты — ещё во время проектирования машины. Например, благодаря им мы не думаем о том – сколько времени нужно, чтобы нагреть воду. "Внутренняя программа" сама активирует нагреватель и выключит его при достижении заданной температуры. Программа также доберет чистой воды при необходимости, закончит стирку и так далее...

В общем, программирование - это процесс написания программ, которые описывают предусмотренный ход событий во времени и порядок правил, которые должны выполниться для достижения запланированного. Окончательный набор таких "ходов и правил" также называют алгоритмом.

А КОМПЬЮТЕРНАЯ программа может быть определена как набор команд для компьютера, составляющие запись алгоритма на одном из языков программирования.

Как мы используем разные языки для общения с людьми из разных стран, разные подмножества языков – жаргоны – для общения на определенные темы типа юрисприденции, медицины, экономики. Как используем разные языки команд бытовой технике, мобильным устройствам... – так и программисты используют разные языки программирования и их "подмножества" – библиотеки – для общения с компьютером с целью его "программирования" на выполнение нужных нам задач и алгоритмов.

В современном компьютерном программировании существует множество направлений в соответствии типам разрабатываемых программ:

- разработка игр - "gamedev";
- разработка мобильных приложений;
- разработка приложений для ПК;
- разработка "встроенных систем" (embedded systems) - вспоминаем стиральную машину ;)
- веб-разработка
- и т.д.

Именно на примере последней "веб-разработки" - мы познакомимся с программированием в следующих занятиях ;)

Веб-разработка?

Возможно, тебе известно, как работать со стандартной программой. Обычно, имея ее установочный файл, мы устанавливаем ее на свой компьютер, открываем и пользуемся. Что же такое веб-сайт? Это такая специальная программа не требующая установки - достаточно просто "открыть" или как говорят "загрузить" ее в Браузере используя ее "адрес". Она может быть открыта одновременно многими пользователями на разных компьютерах и браузерах. Например, тысячи людей могут одновременно использовать сайт Facebook. Он может предоставлять пользователям одну и ту же или различную информацию в зависимости от контекста, позволяя пользователям взаимодействовать как с сайтом, так и с другими пользователями, использующими его. Веб-сайт может состоять из одной или нескольких "веб-страниц", каждая из которых имеет свой "адрес". Эта программа, которую различные пользователи могут загружать через браузер, называется веб-клиентом, и, на самом деле, есть только частью "веб-сайта". Но есть и другая часть "веб-сайта", которую мы не видим, но которая служит его "мозгом", который контролирует одновременную работу всех загруженных пользователем веб-клиентов, отправляет им по запросу нужную информацию, сохраняет свои данные и позволяет им взаимодействовать друг с другом. Этот "мозг" называется веб-сервером.

А такая "умная" веб-программа, состоящая из клиента и сервера - также известна под названием "веб-приложение".

Соответственно, в разработке веб-приложений выделяют два направления:

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

Разработчики, которые умеют и то, и другое (бекенд и фронтенд), называются "фулстек" веб-разработчиками (fullstack web developers). Однако, многие современные проекты настолько сложны, что часто бекенд и фронтенд разработчики - это две отдельные группы специалистов.

Мы начнем с более простой и "более близкой" для конечного пользователя части - фронтенда. На примере разработки веб-приложения для управления задачами - "таск-менеджера" ("task manager") - мы сначала создадим небольшой веб-клиент, с помощью которого пользователь сможет создавать задачи, без возможности сохранить задачи между перезагрузками веб-сайта и без доступности созданных задач с разных компьютеров. А потом - мы возьмемся за разработку веб-сервера с целью сохранить задачи, чтобы они были доступны между перезагрузками веб-сайта, в том числе и с разных компьютеров.

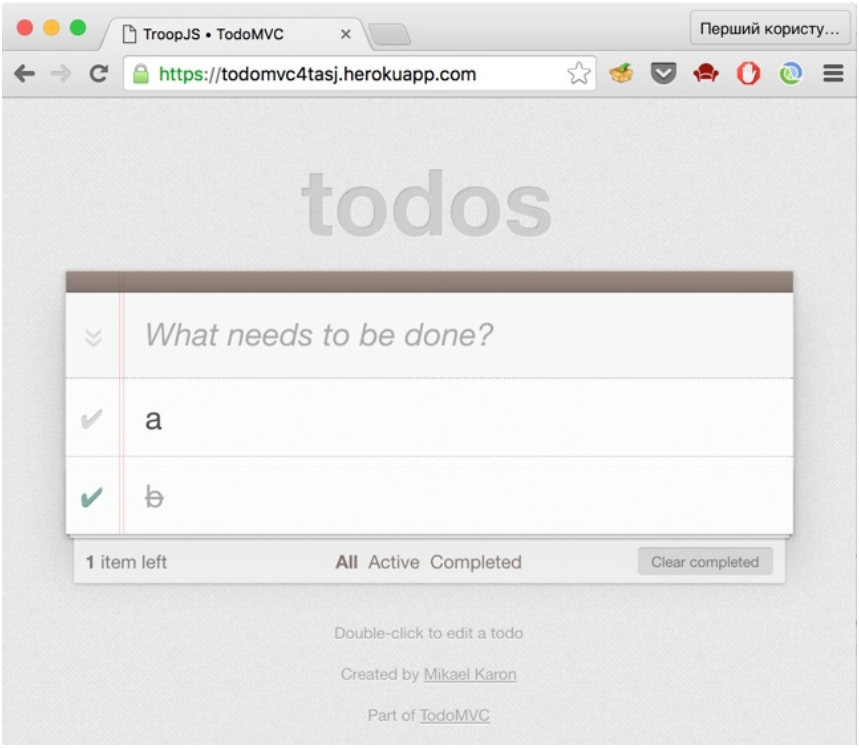
Стоит также отметить, что мы не будем рассматривать все нюансы веб-разработки. Мы не будем всегда следовать точной терминологии, и пользоваться всеми "последними фишками" рассмотренных технологий. При этом мы будем стараться по возможности пользоваться "последними нововведениями", если это упростит подачу материала. Наша цель - не выучить базу программирования за такое ограниченное время, а познакомиться с процессом работы веб-разработчика и почувствовать на практике - насколько это сложно и интересно.

HTML

Веб-страницы - как их видим мы, пользователи

Для того, чтобы начать разрабатывать собственные веб-приложения, мы должны на базовом уровне понимать, как реализованы "веб-страницы".

Мы привыкли называть "веб-страницей" все, что появляется в браузере после загрузки веб-сайта.

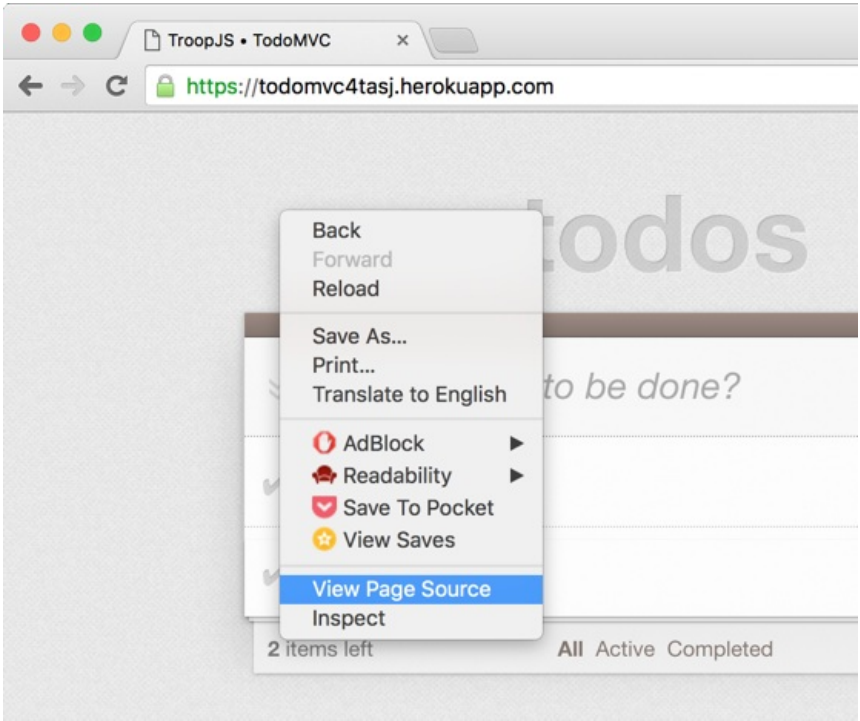


Веб-приложение TodoMVC

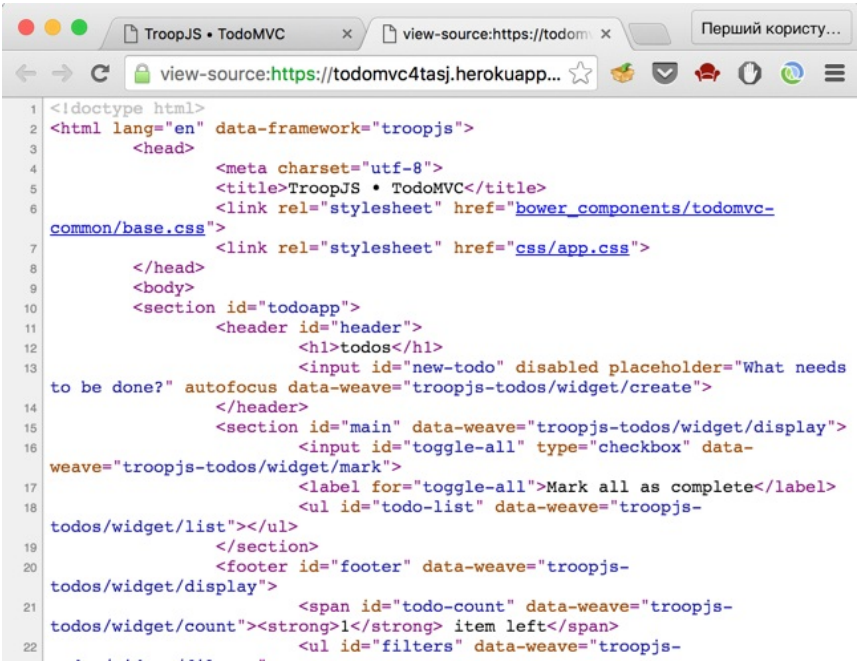
Но за все эти чудесные элементы - поля редактирования, метки с текстом, кнопки, чекбоксы, ссылки - мы должны благодарить браузеры (Firefox, Chrome, Opera, Edge, Safari ...), которые умеют представлять внутреннюю реализацию страниц в форме, понятной для людей. Другими словами, браузеры - это переводчики с "языка программирования веб-приложений" на "язык пользователей". Такой процесс "перевода" также называют "интерпретацией".

Веб-страницы - как их видит браузер

А вот как веб-страница выглядит для браузера:



Выбор "View page source" в контекстном меню страницы



Исходный код страницы TodoMVC

Эта "абракадабра" написана на языке разметки гипертекстовых документов (веб-страниц) - HTML (Hyper Text Markup Language). Следует отметить, что в народе также могут использовать соответствующий англицизм - "аштээмель". Это касается большинства терминов такого типа. И мы далее также можем использовать англицизмы для некоторых терминов, если это будет удобно.

Слово "гипер" означает, что мы имеем дело не только с "линейным" текстом, но и с текстом, который может содержать ссылки (так называемые "гиперссылки") на другие ресурсы. Ссылки могут перенаправлять нас на ту же страницу или на другие страницы с другими адресами.

Чтобы понять секрет этого языка, давай рассмотрим простой пример.

Тебе наверное уже стало понятно, что веб-страница, которую мы загрузили раньше, связана с планированием задач. Так и есть - это уже полноценно реализованный менеджер задач, который позволяет нам создавать задачи, редактировать или удалять их, отмечать задачи как выполненные, фильтровать их или очищать. Фактически, мы получаем полноценную программу в браузере, а не просто веб-страницу с информацией и без возможности динамически изменять ее здесь и сейчас.

Одна из основных целей этого учебника - попрактиковаться создавать подобные веб-приложения самостоятельно.

И сейчас мы приступим к реализации основного функционала нашего менеджера задач. Будем создавать новые задачи, вводя текст в текстовое поле, а после нажатия `Enter` они должны будут появляться в списке, который виден ниже текстового поля.

Вот так мог бы выглядеть базовый HTML-код нашего приложения:

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

Посмотрите внимательно на эту структуру. Можно попробовать себе ее представить как большой шкаф, который начинается с `<html>` и заканчивается на `</html>`. Как мы видим, идентификаторы начала и конца "шкафа" обрамлены специальными символами - "меньше" (`<`) и "больше" (`>`). А чтобы отличать "конец" от "начала" – в случае "конца" - вместо `<` используется `</` .

В середине "шкаф" состоит из двух основных секций - *head* и *body*, которые начинаются с `<head>` , `<body>` и заканчиваются на `</head>` , `</body>` соответственно.

head - играет роль "паспорта" нашей веб-страницы. В том смысле что содержит набор данных описывающих страницу. В настоящее время у него есть только одна "полочка" - название нашей веб-страницы - `<title>Todos</title>` .

А в *body* хранится содержимое нашего сайта со всеми его "элементами-коробками".

Такие элементы-коробки могут быть вложенными одна в другую.

В нашем случае элементы списка

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

который отображает задачи,

вложены в одну большую коробку - `<ul> ... </ul>` , которая содержит список задач.

Кстати, `ul` расшифровывается как "**н**еупорядоченный **с**писок" ("**u**nordered **l**ist").

А `li` - это "Элемент Списка" ("List Item").

Как видим, "коробка" может содержать один только текст - например, элемент списка `<li>watch lesson</li>` .

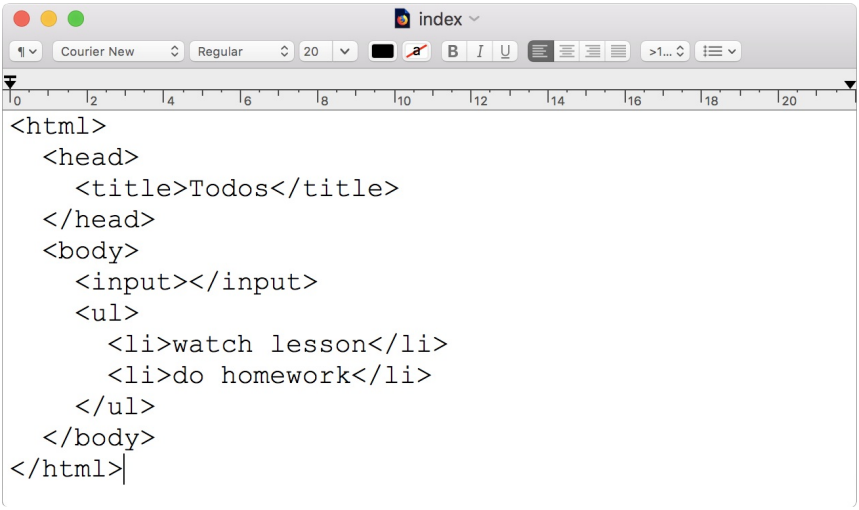
А может не содержать ничего, как, например, поле ввода: `<input> </input>` .

В таких случаях, если коробка пустая, мы можем просто написать `<input/>` .

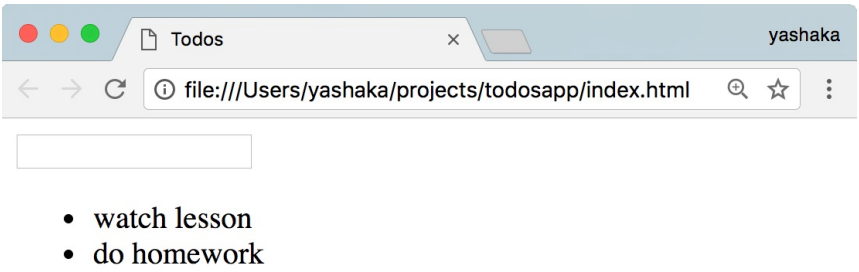
Все эти "коробки" официально называются **элементами**. Начальный идентификатор называется **открывающим тегом** ("opening tag") — например, `<li>` —, а идентификатор конца - **закрывающим тегом** ("closing tag") — например, `</li>` .

Как видишь, все, что делает HTML - это управляет размещением информации о нашем веб-сайте во вложенных "ящиках" с целью организовать ее и отобразить нужную структуру страницы.

Давай теперь посмотрим, как веб-страница с кодом будет выглядеть в браузере. Для этого сохраним этот код в файле с расширением `html` - `index.html` - и откроем его в браузере:



Сохранение html-примера в обычном текстовом редакторе



Открытый html-файл в браузере

Вот и все.

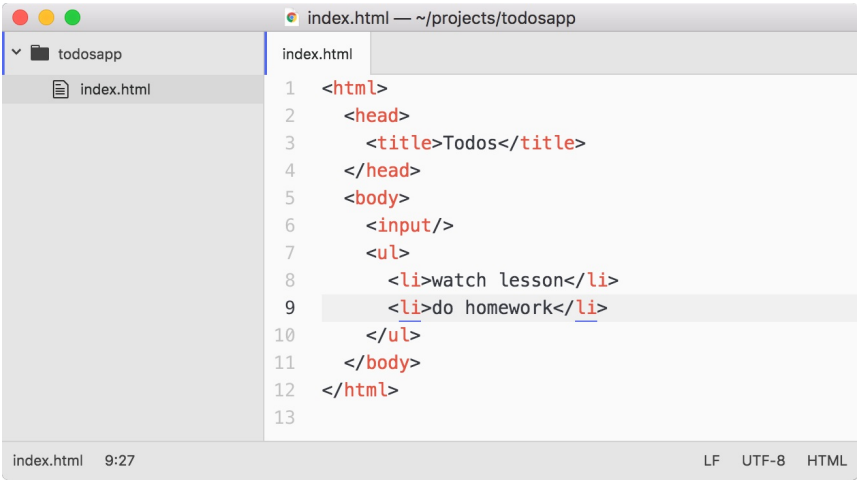
Возможно, ты думаешь - почему именно `index` ? Пусть это будет одним из твоих домашних заданий - раскрой этот секрет самостоятельно ... Google в помощь ;)

Редакторы Кода

Прежде чем продолжить, давай выясним что такое редактор кода.

Любой код, на самом деле, это просто текст, который можно редактировать в обычном текстовом редакторе. Но в коде есть своя специфика по сравнению с обычным текстом. И точно так же, как текстовые редакторы имеют дополнительные функции, например, грамматические подсказки, линейки и т. д., так и редакторы, специализированные для редактирования кода, имеют свои специальные функции, помогающие при работе с кодом.

Вот так выглядит код index.html, открытый в одном из таких редакторов кода - Atom:

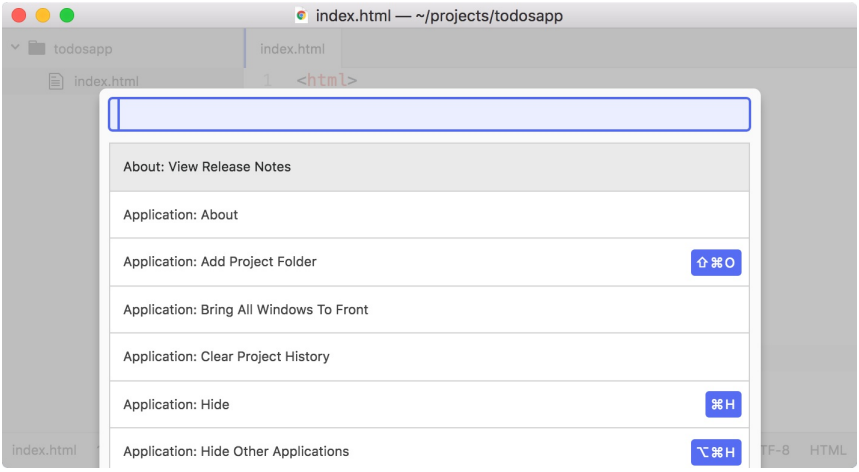


index.html в редакторе Атом

Как видишь, сразу читать такой код стало легче из-за особой подсветки синтаксиса. Теперь в глазах все не так расплывается из-за "простого текста", смешанного с "тегами элементов".

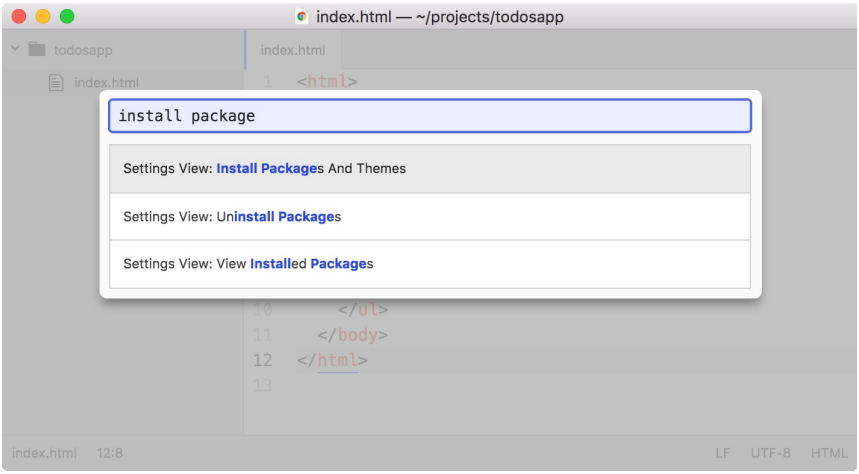
И еще кое-что интересное об "Атоме". Существует множество дополнительных "модулей" или так званных "пакетов" ("packages"), которые облегчают "будни программиста", добавляя к базовому функционалу редактора дополнительные функции. Например, было бы здорово иметь непрерывный "живой просмотр в браузере" нашего кода - возможность наблюдать в эмуляторе браузера, как наш код будет в итоге выглядеть после каких-то изменений. Есть ли такой пакет для "Atom"? Не знаю, давай искать;)

Cmd-shift-p для Mac или *ctrl-shift-p* для Windows откроет диалог "Command Palette", в котором мы можем найти и выполнить любую фичу (от "feature" - определенная часть функционала) редактора:



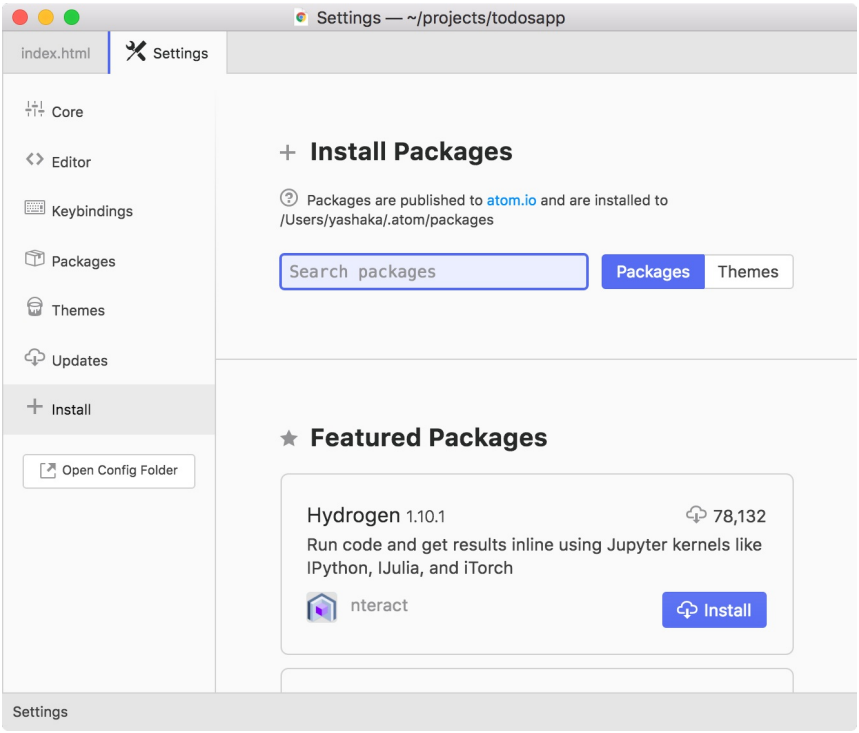
Command Palette

Давай поищем "install package" ("установить пакет"):



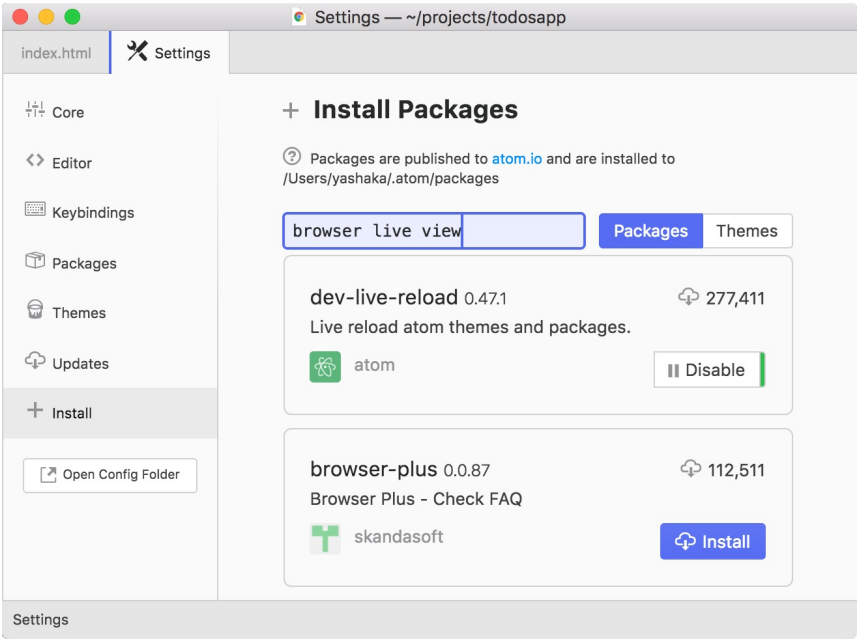
Поиск "install package"

Отлично, похоже, что "Settings View: Install Packages And Themes" - это как раз то место, где мы можем проверить, какие пакеты мы можем установить для "Atom":



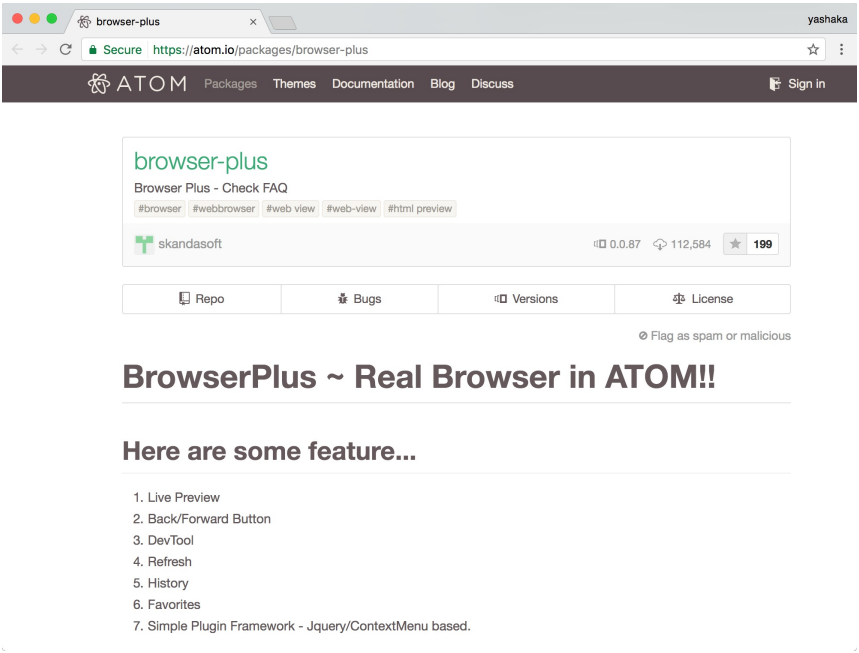
Settings>Install

Теперь перейдем к "Search packages" и поищем "browser live view" ("живой просмотр браузера"):



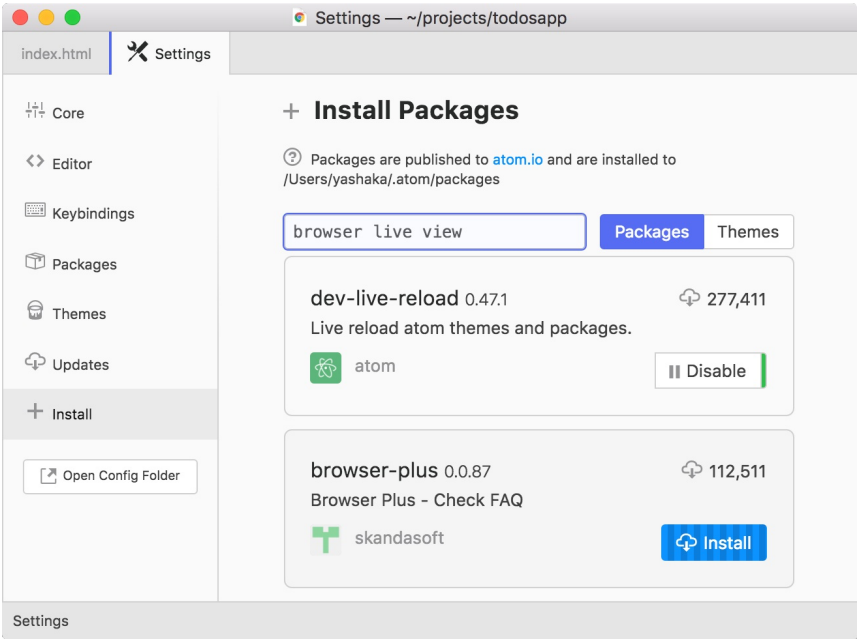
Поиск "browser live view"

О! Второй пункт в списке - "browser-plus". Если кликнуть по его названию, мы получим страницу с подробной информацией о пакете:

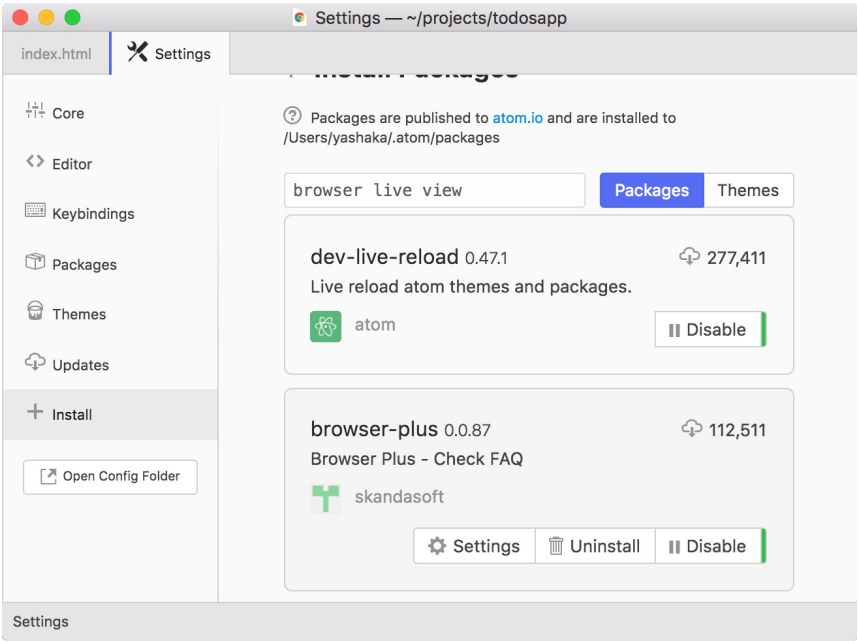


Информация о browser-plus

"Real Browser in ATOM" ("Настоящий Браузер в Атом")! И первая его особенность - "1. Live Preview". Думаю, это то, что нам нужно. Давай установим этот пакет.

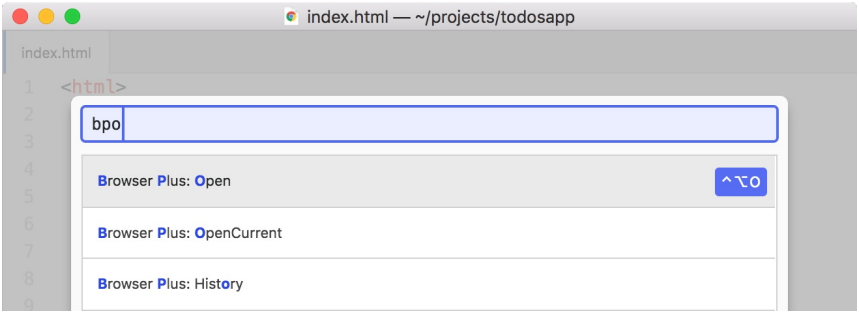


Установка пакета "browser-plus"

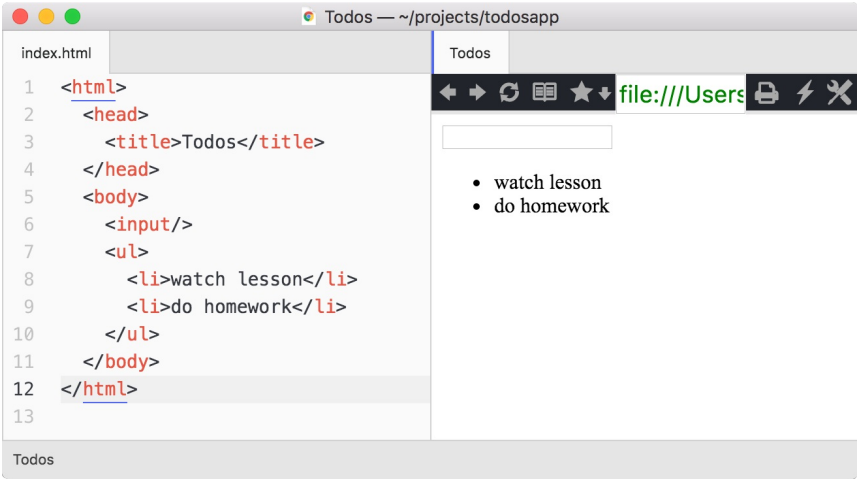


Установленный пакет "browser-plus"

Теперь пакет "browser-plus" установлен, и мы можем включить его, используя уже известную нам "Command Palette" (обрати внимание, что мы можем искать необходимую команду даже по первым буквам слов из ее названия):

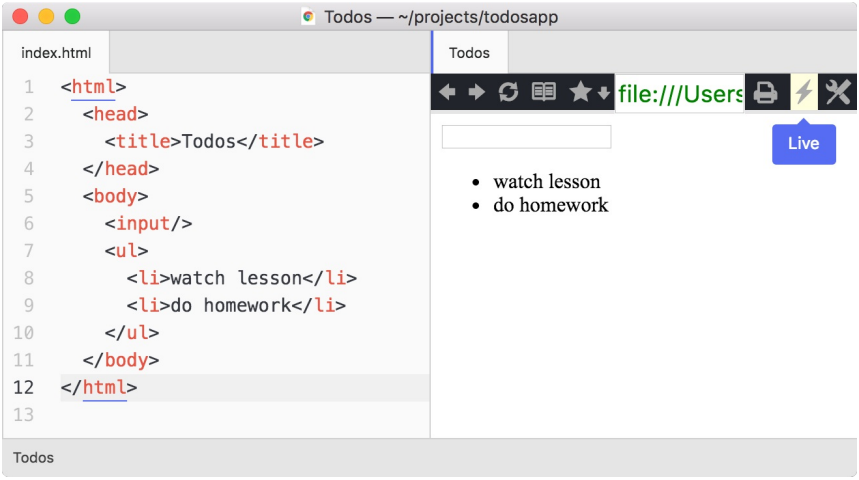


Включение "browser plus"



Включенный "browser plus"

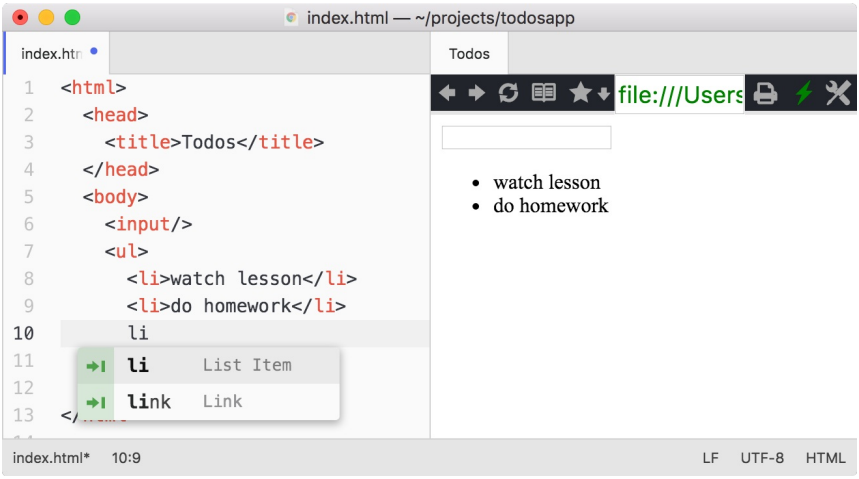
Чтобы включить функцию "живого просмотра", нужно нажать соответствующую кнопку с изображением молнии:



Включение "live preview"

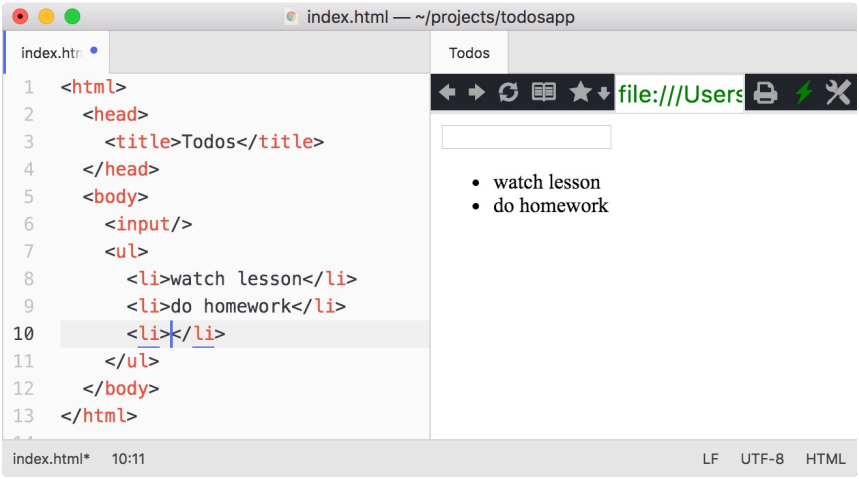
И теперь, если сохранить любое новое изменение в файле (через *cmd+s* на Mac или *ctrl+s* в Windows), изменения сразу будут отображены в эмуляторе браузера.

Давай, например, добавим еще одну задачу в список. Здесь мы можем познакомиться с еще одной особенностью "Atom" - автозаполнением. Давай начнем добавлять новый элемент `li`, просто введя символы `l` и `i` (без символов тегов - `‹` и `›`):



Вызов подсказок автозаполнения

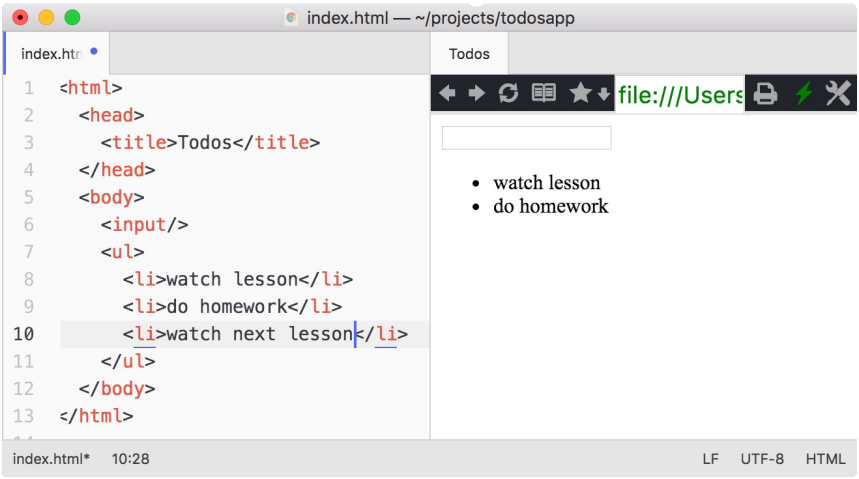
А теперь нажмем `Tab` или `Enter` , чтобы завершить процесс "автозаполнения":



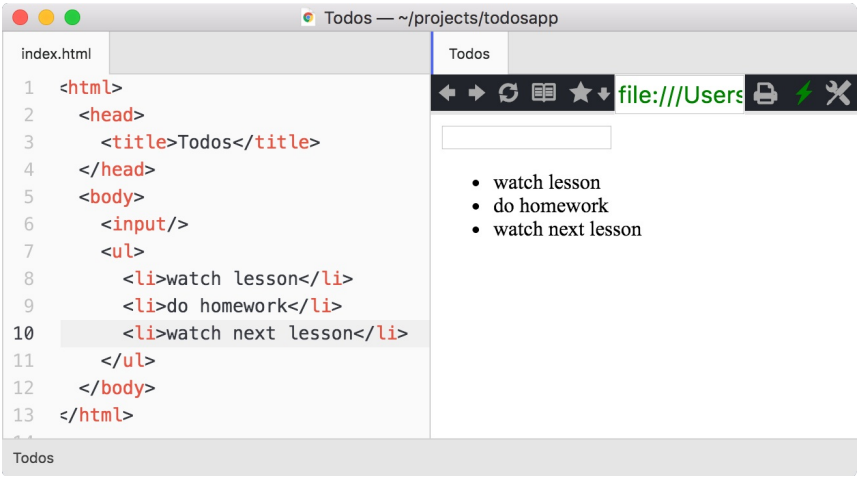
Завершение процесса автозаполнения

Редактор сам добавит символы открывающего тега (‹ и ›), а также автоматически добавит закрывающий тег.

Теперь, добавив текст в новую задачу и нажав *cmd+s* на Mac или *ctrl+s* в Windows (сохранение изменений), мы увидим, что они будут отображены и в эмуляторе браузера:



Изменение перед сохранением



Отображение изменений после сохранения

Круто, не правда ли? :)

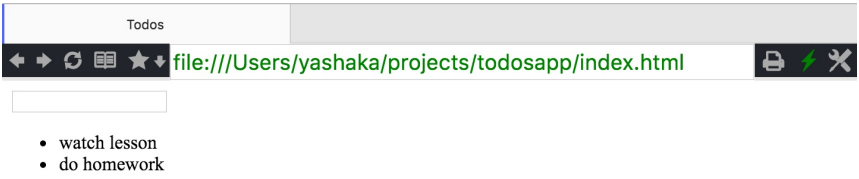
HTML Атрибуты

Не для всех знакомых нам элементов веб-страниц существуют свои специфические теги.

Например, большинство элементов, которые предполагают "фидбек от пользователя" - например, ввод текста, выбор чекбокса или радиокнопки, нажатие кнопки - могут быть реализованы с помощью элемента с тегом `input` .

Раньше мы уже использовали элемент `input` для представления поля редактирования, в котором пользователь может вводить текст новой задачи.

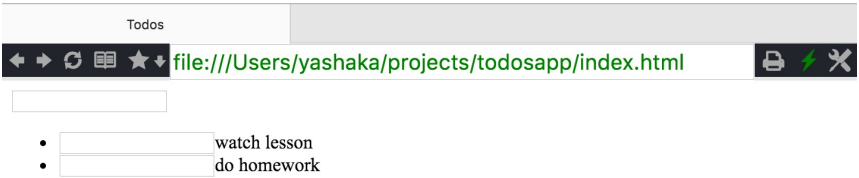
```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input />
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```



Элемент `input` как текстовое поле

Просто для примера, давай теперь добавим "чекбоксы" ("checkboxes") для каждой задачи, чтобы дать возможность их завершить или отметить как "сделанные":

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



Элементы input для завершения задач

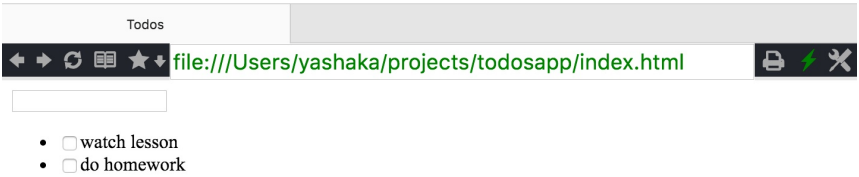
Но как теперь указать, что новодобавленные элементы `input` для задач должны быть чекбоксами?

В таких и других ситуациях, когда нам нужно предоставить дополнительную информацию об элементе или, иначе говоря, добавить "индивидуальности" элементам, — используются **html-атрибуты**.

Атрибуты бывают "общими", которые можно добавить к любому элементу, а бывают "специфическими только для определенных элементов", то есть имеет смысл добавлять их только к элементам определенных тегов. Это касается, например,

атрибута `type` который актуальный именно для элемента `input` . В нашем случае атрибут `type` позволяет нам точно обозначить элемент `input` именно как 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>
```

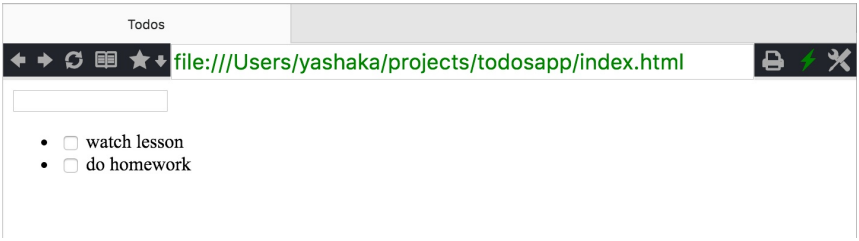


Элементы `input` как чекбоксы

Только для лучшей структурности, чтобы более точно выделить каждую часть функциональности задачи - давай поместим текст наших задач в свои собственные элементы играющие роль "этикеток", "меток" или "надписей" для задач - элементы `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>
```



Элементы label

Теперь функциональность "завершения задачи" и функциональность "отображения текста задачи" отражаются в разметке отдельными элементами. В будущем это позволит нам более удобно и точно находить "необходимую функциональность" в коде. Идея точно такая же как и "разложить по полочкам в шкафу раскиданные вещи";)

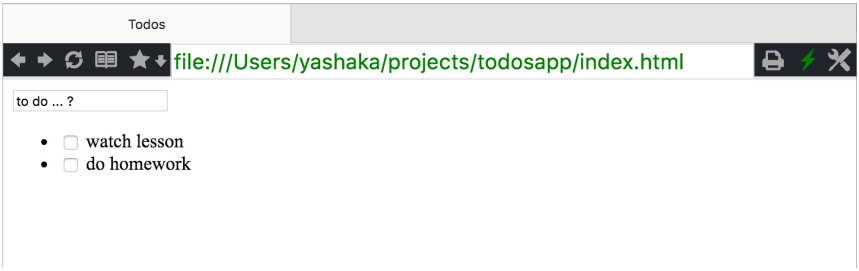
Давай познакомимся с некоторыми другими атрибутами "функционального" типа, которые добавляют важные "функции" нашим элементам.

Как тебе идея добавить текст подсказки для пользователя в текстовое поле?

Вот как мы можем это сделать, используя атрибут `value` элемента `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>
```



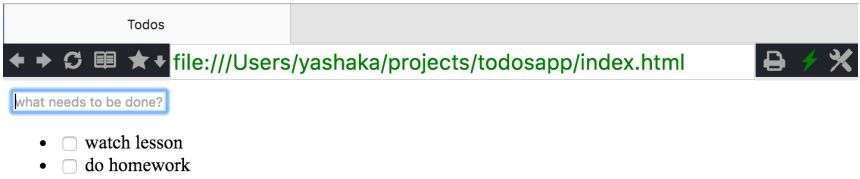
Элемент `input` с атрибутом `value`

Работает и "обратная связь" - если мы введем текст в поле редактирования в браузере, то этот текст будет сохранен в атрибуте `value`. Хотя, мы еще и не знаем, как это проверить. Но мы вернемся к этому чуть позже – когда используем эту особенность, чтобы "подсмотреть" значение атрибута `value` текстового поля после нажатия пользователем `Enter`, а затем добавим новую задачу с этим подсмотренным текстом в список ;)

Минуточку, но ведь если мы введем текст в текстовое поле, то пользователь будет вынужден каждый раз его сначала удалять, а потом уже вводить свой...

Есть простой способ это исправить. Оказывается, есть еще один атрибут - `placeholder` , который используется как "подсказка для пользователя", и не мешает вводу нового текста:

```
<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>
```



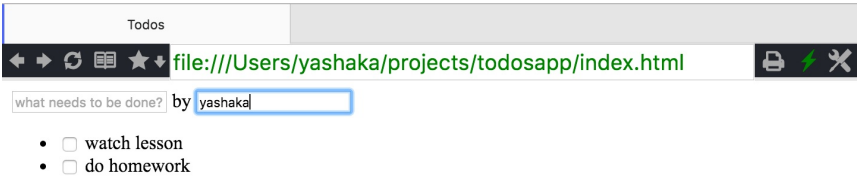
Элемент `input` с атрибутом `placeholder`

Итак, с помощью атрибутов мы можем предоставить Браузеру дополнительную информацию об элементе. Браузер умеет интерпретировать специфичный набор атрибутов элемента и их значений и изменять отображение элемента на странице

соответственно. Например, если у элемента `input` есть атрибут `type="checkbox"` , то Браузер отобразит его как чекбокс. Но не для всех подобных наших пожеланий о изменении стиля элемента или определенного поведения, связанного с ним, существуют заготовленные атрибуты и их значения. Вот например, нам бы не мешало изменить стиль элемента `input` для ввода текста новой задачи так, что-бы он отображался по центру, а главное - научить его реагировать на нажатие `Enter` создавая новую задачу с введенным ранее текстом. Для этого прийдется написать отдельный код в отдельних файлах на других языках, который будет сам находить нужные элементы и изменять их стиль и поведение. Мы займемся этим чуть позже, а сейчас давай поразмыслим вот над чем. Как такой код сможет быстро найти именно наш элемент "input" среди других элементов "input"? Как отличить элемент `input` как "текстовое поле" от элемента `input` который чекбокс?

Мы могли бы научить этот код искать *"элемент input, который не является чекбоксом"*, но что делать, если у нас есть другое текстовое поле, которое отображает имя пользователя, назначенного для этой задачи?

```
<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 текстовых поля

Как отличить два текстовых поля в таком случае? Неплохо было бы использовать поиск вида - *"найдите элемент input с атрибутом placeholder со значением 'What needs to be done'"*.

Но что, если наша веб-страница поддерживает 10 языков? Не будет ли трудно перечислять все варианты?

> *"Найдите элемент input с атрибутом placeholder со значением 'What needs to be done?' или 'O que precisa ser feito?', или 'Що потрібно зробити?' или '需要做什麼呢' или ... "*.

И надо заметить, что в будущем наша веб-страница может стать более сложной и, вероятно, будут добавлены новые элементы `input`. Все это усложнит поиск нужного элемента нашим дополнительным кодом для "добавления задачи, нажав `Enter`".

Подводя итог, поскольку мы можем иметь элементы одного и того же типа (т. е. с одним и тем же тегом), но для разных целей, - нам нужен четкий способ маркировки элементов для того, чтоб их различать.

Для этой цели существуют специальные атрибуты:

- Атрибут `id`, который присваивается уникальным элементам веб-страницы;
- Атрибут `class`, который присваивается элементам, принадлежащим к определенной группе.

Благодаря таким атрибутам мы можем связать необходимый функционал с соответствующими элементами в рамках дополнительного кода, упомянутого ранее, который нам предстоит реализовать.

Итак, давай пометим наши элементы `input` в соответствии с их ролями:

```
<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>
```

Теперь у нас всегда есть четкий способ отличить одно текстовое поле от другого. И даже "сделать выборку" группы чекбоксов, соответствующих классу "toggle". Это может быть полезно, чтобы придать им специфический уникальный стиль.

Именно тюнингом стиля нашего приложения мы и займемся в следующей главе;)

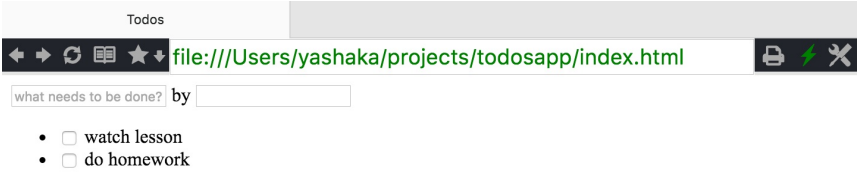
Добавляем уникальный стиль (CSS)

Добавление пользовательских стилей в html-страницу

HTML дал нам возможность структурировать содержимое страницы - чтобы разметить ее данные, отобразив их иерархическую вложенную 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 умеет визуально представлять эти "структурированные данные" в соответствии со "стилями по умолчанию".



Стиль по умолчанию

Конечно, было бы здорово настроить стиль визуального представления данных веб-страниц так, как нам хочется.

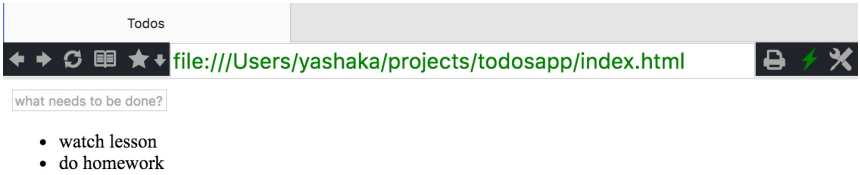
Это можно сделать с помощью другого инструмента веб-разработки - **Каскадных Таблиц Стилей** (Cascading Style Sheets) или коротко **CSS** (или используя популярный англицизм – "ЦСС").

Это специальный язык, который позволяет нам описывать стилистические свойства соответствующих элементов в виде набора правил.

Обычно стили описываются в отдельных файлах с расширением `.css` .

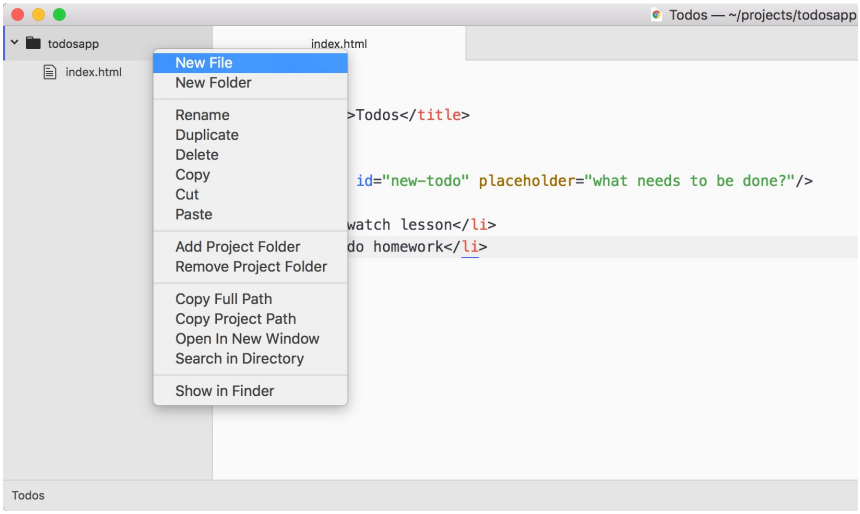
Перед тем, как написать стили для нашего веб-приложения, давай упростим наш код HTML – до той структуры, которая необходима для реализации функционала только "добавления задач". Это еще и облегчит понимание всего процесса.

```
<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>
```

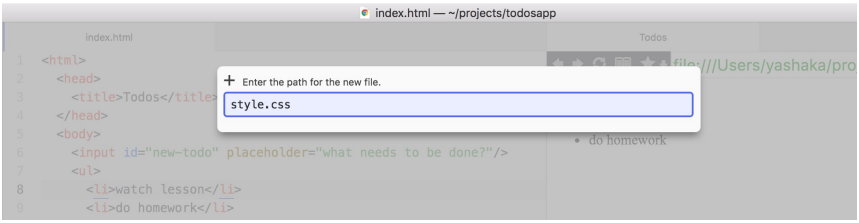


Упрощенный код в браузере

Теперь в рамках нашего проекта (мы можем включать/выключать панель с "деревом" проекта, так званным "tree view", — с помощью "cmnd+\" для Mac или "ctrl+\" для Windows) создадим новый файл `style.css` :



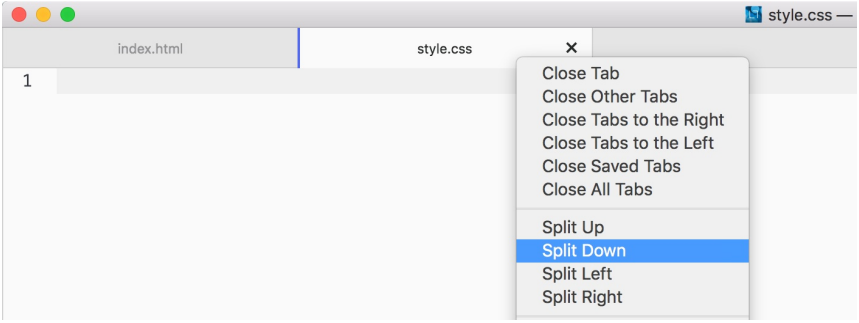
Создание нового файла через контекстное меню с панели дерева проекта



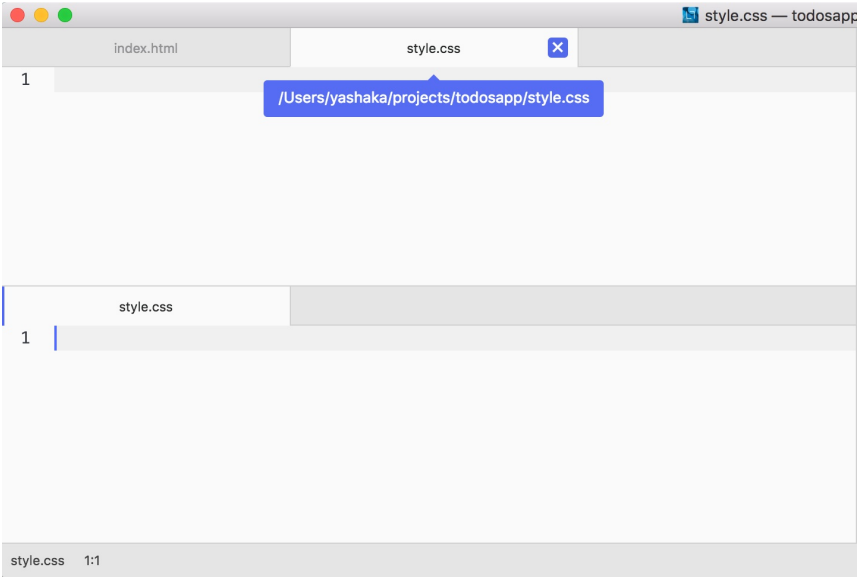
Задание имени нового файла

Созданный новый файл

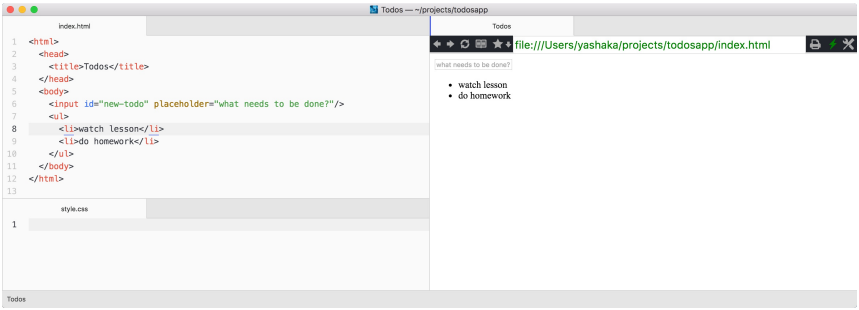
Для удобства давай закроем левую панель со структурой проекта ("cmnd+\" для Mac или "ctrl+\" для Windows) и переместим открытую новую вкладку в отдельную секцию в редакторе кода с помощью элемента контекстного меню "splitting the window down":



закроем "дубликат":



и, наконец-то, подправим размер вкладок в соответствии с нашим вкусом:



Мы еще не написали ни одного "правила стиля", но давай сразу подключим наш файл "css" к html-странице, чтобы встроенный в редактор браузер мог сразу их "применять" по ходу написания и сохранения нового кода.

"Подключение" состоит в том, чтобы добавить специальную "регистровую запись"

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

в "паспорт нашей страницы" - секцию `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 что ж, давай "украшать" наш сайт :)

И начнем с текстового поля - тебе не кажется, что оно слишком короткое?

Давай "растянем" его на всю ширину страницы.

Сначала сформулируем **правило стиля** на обычном русском языке:

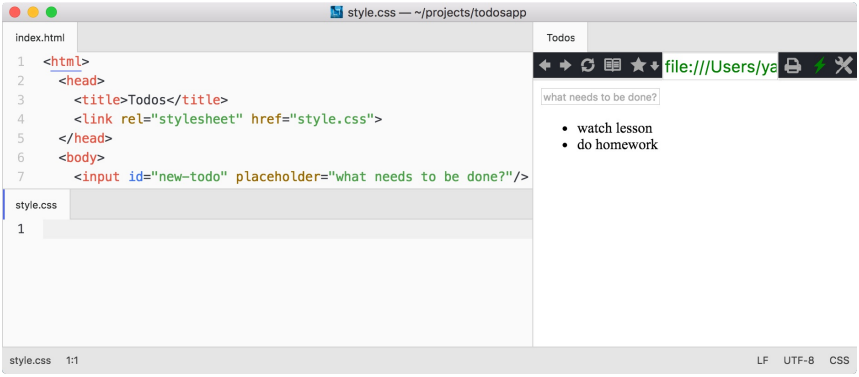
> Элемент с `id="new-todo"` должен иметь ширину во все доступное пространство

Или более точным "техническим" языком:

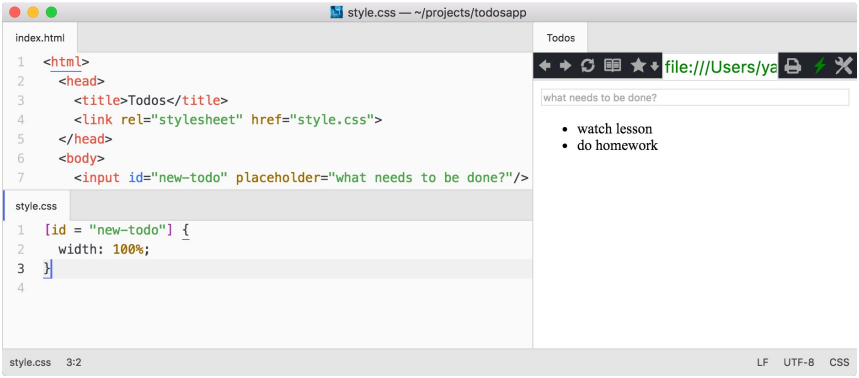
> Элемент с `id="new-todo"` должен иметь ширину в 100% доступного пространства

А вот и перевод на "язык CSS":

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



Элемент input с шириной по умолчанию



Элемент input с шириной в 100%

Как видишь, перевод достаточно однозначный:

Элемент с id="new-todo" ...

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

... должен иметь ...

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

... ширину в 100% доступного пространства

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

CSS-селекторы

Правило начинается с **селектора**, и он определяет выборку элементов, к которым будет применено это правило:

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

В этом случае - элементы, которые имеют атрибут `id="new-todo"` (у нас таких - ровно один, для этого мы ранее и использовали атрибут для уникальной идентификации элемента - `id`).

Синтаксис квадратных скобок

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

это универсальный способ сказать: *"элемент или элементы, у которых атрибут имеет такое-то значение"*. Например, мы могли бы "найти наш элемент" и через **выборку по двум атрибутам**:

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

```
}
```

Также селектор позволяет отличать "поиск по атрибутам" от **"поиска по тегу элемента"**:

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

Теперь наш селектор говорит:

> *"найди элемент(ы)* ***с тегом input****, с атрибутом* `id="new-todo"` *, и атрибутом* `placeholder="What needs to be done?"`

В любом случае, поскольку `id` - специальный атрибут, уникальный для элемента на странице, мы можем ограничить поиск только одним атрибутом - `id` :

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

Более того, оказывается, что люди так часто ищут элементы по атрибуту `id` , что в CSS предусмотрели сокращенный синтаксис:

вместо

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

можно писать просто

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

Свойства CSS

Давай продолжим разбирать синтаксис CSS-правил...

Элемент с `id="new-todo"` должен иметь ширину в 100% доступного пространства

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

После селектора - в фигурных скобках - следует **блок определения правила**:

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

в котором перечислены "стилистические" **свойства**, которые необходимо "установить" для найденных элементов или элемента:

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

Свойства определяются в соответствии со следующим синтаксисом:

имя свойства

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

двоеточие

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

значение

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

точка с запятой

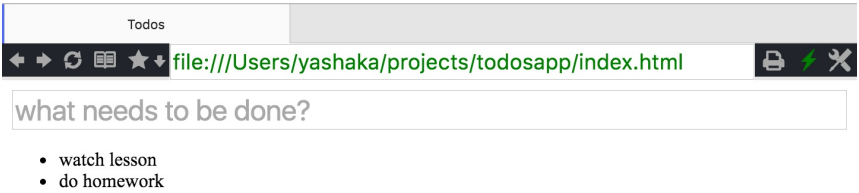
```
#new-todo {
     width: 100%;
}
```

Больше примеров. Настраиваем стиль шрифта

Давай добавим в список больше свойств... ;)

Увеличим размер шрифта до 24 пикселей:

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



Сделаем шрифт курсивом:

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