Intro to Software Development

Iakiv Kramarenko

Table of Contents

Intro to Software Development

Iakiv Kramarenko

This book is an early access to an "Intro to Software Development" course. It gives an introduction to programming by examples in Web Development using HTML, CSS, and JavaScript. The book should be on strength for all — from kids to their parents, with the only prerequisite to be a confident computer user. It should help to taste the development of a small but real product and determine your desired role in IT - developer, tester, or maybe somebody else;)

The book is available for free download and for donation based sale at http://leanpub.com/intro-to-software-development.

Cover photo by Rafael Zamora.

This version was published on 2018-08-07.

Foreword

In 2015, I started to give paid offline and online IT courses (programming, test automation, etc). I noticed a few things. Firstly, the majority of available courses on the market, especially free ones, were too technical and complicated for students who start their way in IT from the very beginning. Secondly, it was hard to decide which way to choose in IT - management, business analysis, design, development, testing, etc. That was the time I started to think on some course to give an introduction to the whole Software Development process, be free, and be on strength for almost anybody - from kids to their parents without prior deep knowledge in Information Technologies, with the only prerequisite - to be just a confident computer user.

The idea was to create a course through each a student can build a real application from scratch. Where each course lesson would represent one stage of the complete Software Development process. As defined by en.wikipedia.org, Software Development —

> is the process of conceiving, specifying, designing, programming, documenting, testing, and bug fixing involved in creating and maintaining applications, frameworks, or other software components.

I started to work on this course in 2016. The following lessons were thought to be included in the course:

- Process
- Business Analysis
- Design
- Frontend Development
- Backend Development
- Test Automation
- Testing
- Deployment

A student was supposed to be introduced to each stage of the process by examples of building a real web application from scratch - a tasks manager. Where each lesson shows how to plan, analyze, design, develop and test basic features of the tasks manager, and then through available exercises per lesson, a student would practice in extending the functionality of the tasks manager by his own, with the help of available tips and tricks, frequent questions and answers.

With the time I understood that the scope of the work to be done is tremendous. Especially taking into account my load on other projects. Till now I have written a draft of the "Process" lesson and completed the Frontend lesson without exercises. Probably I will publish the draft of the "Process" lesson as a blog post. And this book, at least in the begininng, is supposed to be a home for the "more programming-like" materials of the course in the book-like format (the contents can change):

- Frontend Development (HTML, CSS, JavaSript)
- Quality Assurance Practices. Automation
- Deployment
- Backend Development
- Testing

The Frontend part is already available (without exercise). I plan to keep the book available for free access and download at all times. But the progress of developing the next lessons, and finally creating a complete course based on the book will depend on donations. The more donations I collect, the less time I will have to spend on my other commercial projects, and so have more time to work on this book and the course. Feel free to donate through the book page at leanpub (http://leanpub.com/intro-to-software-development) or by sending coins to my wallets:

Bitcoin - 1EyDGuW64YkJbZ8FW1yAkz6iLP8c6tCjn

Bitcoin Cash - qqp2g49z0eskfv5kyv53q4rf2huz8lqssqe47ysmyr

Ether - 0x7f2cAa79D1f1966d3CDd8295f8aF6028D66de00e

Programming?

"Programming" - a word we should all be familiar with. We all use home appliances, which we program to perform needed actions in needed order. For example, we build a washing program for washing machine: set timing, intensity, extra rinse, etc. Such different configurations are the simplest forms of algorithms but we already can call ourselves - "programmers". We even program our children, when educating them.

Computer programmers - do the same but on a much lower level.

Back to the washing machine. Who created that customizable washing profiles? That was the programmers' job on the stage of building machine. For example, thanks to them we do not think about how long it takes to heat the water. "Internal program" itself enable the heater and turn it off when the specified temperature is reached. The program will also change the water when needed, stop washing, and so on ...

So, in short, Programming - is the process of writing programs that describe predefined flow of events in time and order of rules that must be satisfied to perform a specific task. The final set of such "flows and rules" that defines a specific task or sub-task is also called an algorithm.

And COMPUTER program can be defined as a collection of instructions or commands) for a computer, that record algorithms with one of programming languages.

In our life we use different languages to communicate with citizens of different countries. Also we use different language subsets like slangs to communicate on jurisprudence, medicine or economics. We use different commands to home appliances, mobile devices, etc. In the same way programmers use different programming languages and their "additional subsets" - libraries - to communicate with a computer.

In modern computer programming, there are many areas defined by the type of programs:

- Game Development
- Mobile Application Development
- Desktop Application Development
- Development of "Embedded Systems" - remember the washing machine?;)

- Web Development
- etc.

In the following chapters we are going to get familiar with programming on the the example of "Web Development" ;)

Web Development?

Perhaps you know how to work with the standard program. Typically, you download it from the Internet, install on your computer, open, and use it. What is a website? For us, it is primárily a special program that does not require an installation. It's enough just to "open" it in Browser by its "address", or in other words - "load" it in Browser. It can be opened simultaneously by many users on different computers and browsers. For example, thousands of people can use Facebook site simultaneously. It can give users the same or different information depending on context, let users interact both with the program and with other users which use the site. The website may consist of one or many "web pages", each of which has its own "address". This program, that various users can "load" through Browser - is called a web-client and, in fact, is only a "part" of a website. But there is the other part of the "website" that we do not see, but that serves as the "brain" of the site that controls the simultaneous operation of all user-downloaded web-clients, sends them by request the right information, stores their data and lets them interact with each other. This "brain" is called a web server.

This "intelligent" web site that consists of a client and a server - is called a "web application".

Accordingly, there are two sub-areas in the development of web applications:

- Frontend Web Development (of client)
- Backend Web Development (of server)

Developers who do both (backend and frontend) are called "full stack" web developers. However, due to the complexity of many of today's projects often backend and frontend developers are two separate branches of specialists.

Let's start with a simpler and "closer" to the end user part – Frontend. We will create a small web-client through which a user can create tasks. Later we will develop a web-server in order to implement functionality that does not belong to Frontend. Such as saving tasks between web site reloads and the ability to see created tasks from different computers.

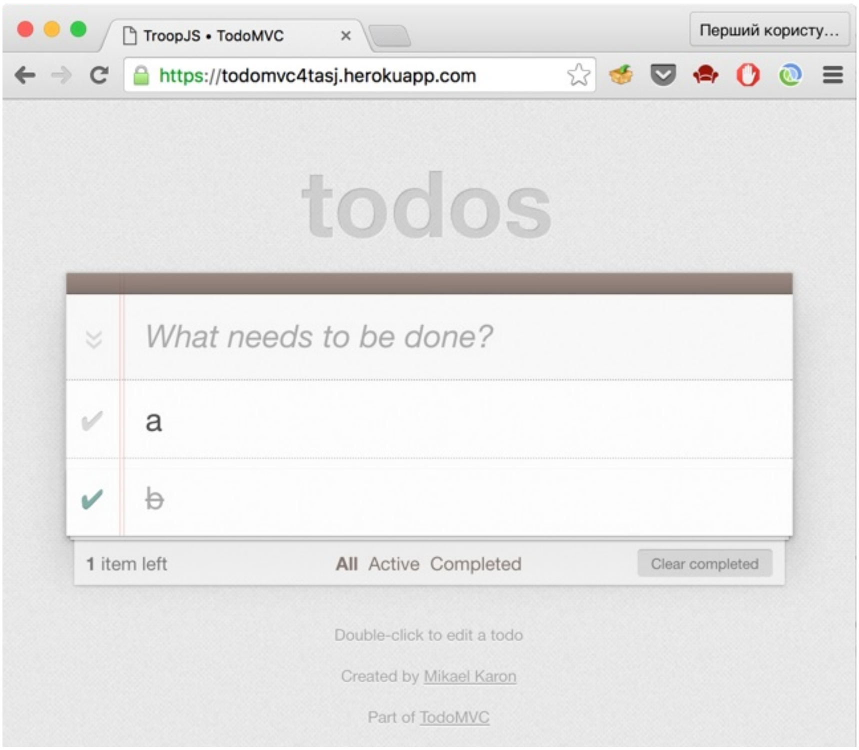
It should be also noticed that we will not touch all the nuances of web development in this course. We will not always follow the accurate terminology and will not use all the most recent innovations. But we will use something from the "new world" if this simplifies the explanation. Our goal - is not to study all programming for a limited time but get familiar with the process of web development, learn how to program something real, and realize in practice - how interesting it may be.

HTML

Web pages - as we, users, see them

At the beginning of developing our own web applications, we need to understand at a basic level how "web pages" are implemented.

We are used to calling a "web page" everything that appears in Browser once a website is loaded.

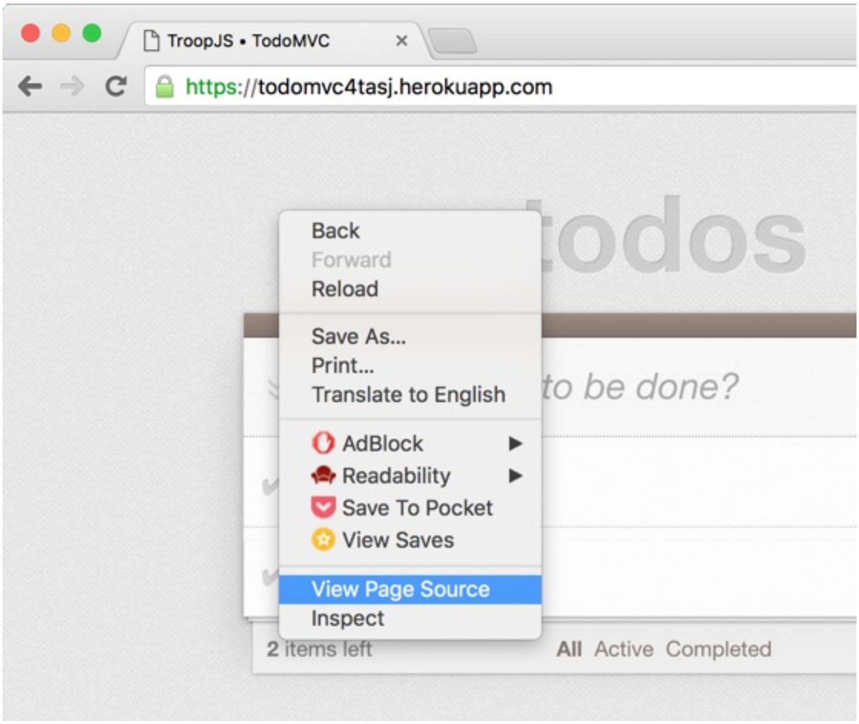


TodoMVC web application

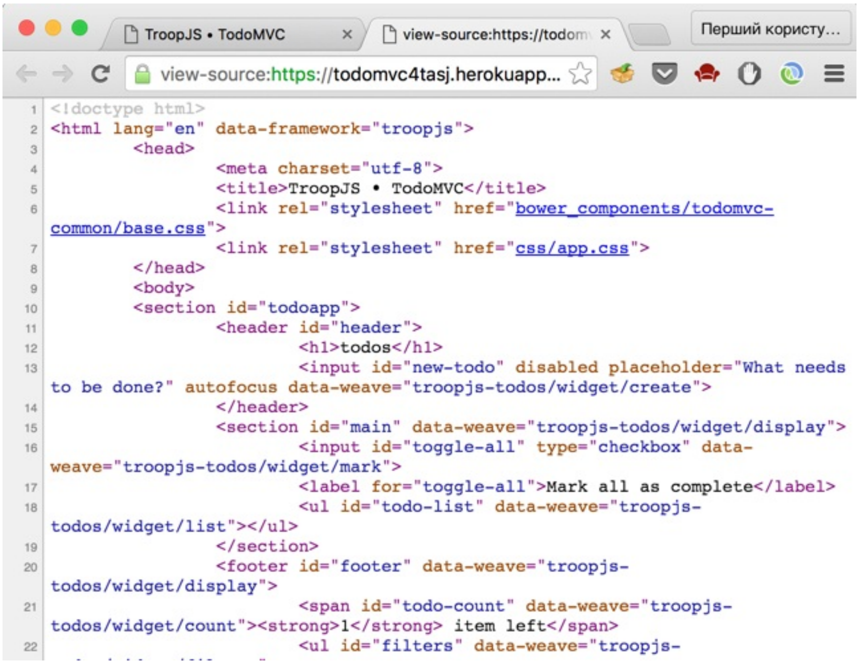
But for all these cute elements - edit fields, labels with text, buttons, checkboxes, links – we have to thank browsers (Firefox, Chrome, Opera, Edge, Safari ...), which can represent the inner implementation of pages in a clear to people form. In other words, browsers are translators from "web application programming language" to "users language". Such "translation" process may also be called "interpretation".

Web pages - as a browser sees them

And here is how a web page looks like for the browser:



Selecting "View page source" from the page context menu


TroopJS • TodoMVC
view-source:https://todom
Перший користу...
view-source:https://todomvc4tasj.herokuapp...

```
<!doctype html>
<html lang="en" data-framework="troopjs">
        <head>
                <meta charset="utf-8">
                <title>TroopJS • TodoMVC</title>
                <link rel="stylesheet" href="bower_components/todomvc-common/base.css">
                <link rel="stylesheet" href="css/app.css">
        </head>
        <body>
        <section id="todoapp">
                <header id="header">
                        <h1>todos</h1>
                        <input id="new-todo" disabled placeholder="What needs to be done?" autofocus data-weave="troopjs-todos/widget/create">
                </header>
                <section id="main" data-weave="troopjs-todos/widget/display">
                        <input id="toggle-all" type="checkbox" data-weave="troopjs-todos/widget/mark">
                        <label for="toggle-all">Mark all as complete</label>
                        <ul id="todo-list" data-weave="troopjs-todos/widget/list"></ul>
                </section>
                <footer id="footer" data-weave="troopjs-todos/widget/display">
                        <span id="todo-count" data-weave="troopjs-todos/widget/count"><strong>1</strong> item left</span>
                        <ul id="filters" data-weave="troopjs-
```

TodoMVC page source

This "abracadabra" is written in Hyper Text Markup Language also known as HTML.

The word "hyper" means that we deal not just with the "linear" text, but with the text that may contain links (called "hyperlinks") to other resources. Links may direct us to the same page or to other pages with other addresses as well.

In order to understand the secret of this language, let's take a look on a simple example.

Probably, you've already paid attention that the web page, we loaded earlier, was related to tasks. It is the task manager that allows us to create tasks, edit or delete them, mark tasks as "done", filter them or clear. In fact, we get a full application in the browser, not just a web page with information without the possibility to dynamically change it here and now.

One of the main goals of this tutorial is to practice in creation of similar web applications by yourself.

And now we are going to start implementing our task manager main functionality - creating new tasks by entering their text in a text field and after pressing Enter - display them in a list below.

That's how a basic HTML code of our application might look like:

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

Look carefully at this structure. It is like a large cupboard that starts from `<html>` and ends with `</html>` . As we see - identifiers of the beginning and the end of the "cupboard" - are framed by special symbols - "less than" (`<`) and "greater than" (`>`). And in order to distinguish the "end" from the "beginning" - the `</` symbols are used instead of `<` in the "end" case.

In the middle, "cupboard" has two main sections - *head* and *body*, which start with `<head>` , `<body>` and end with `</head>` , `</body>` correspondingly.

head - acts as an "ID card" of our web page. In the sense that identifies the page with a set of information about it. It has currently only one "shelf" - the name of our web page - `<title>Todos</ title>` .

Body stores the content of our website with all its elements-boxes.

Such elements-boxes can be nested.

In our case, list items

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

that reflect tasks,

are nested in a single big box - `<ul> ... </ul>` - which displays a list of tasks.

By the way, `ul` is deciphered as "**u**nordered **l**ist".

And `li` - as "**l**ist **i**tem"

As we can see, the "box" may contain only text - such as list item `<li>watch lesson</li>`

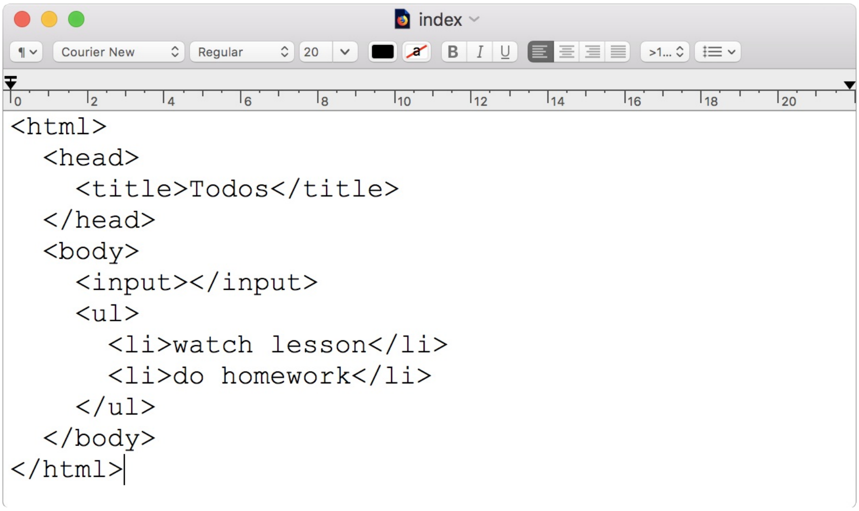
And may contain nothing, as the input field: `<input> </input>`

In such cases, when the box contain nothing, we can just write `<input />` .

All these "boxes" are officially called elements. Beginning identifier is called an **opening tag** (e.g. `<li>`), and the ending identifier - **closing tag** (e.g. `</li>`)

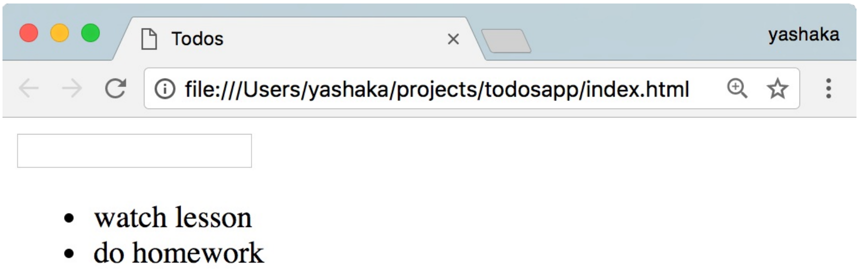
As you can see, all that makes HTML – are rules of placing information about our website in nested "boxes" in order to organise it and reflect the corresponding structure of a page.

Let's take a look how the web page with the code will look like in a browser. To do this, let's keep this code in a file with the extension html: index.html and open it in the browser:



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

Saving html example in common text editor



- watch lesson
- do homework

Opened html-file in the browser

That's it.

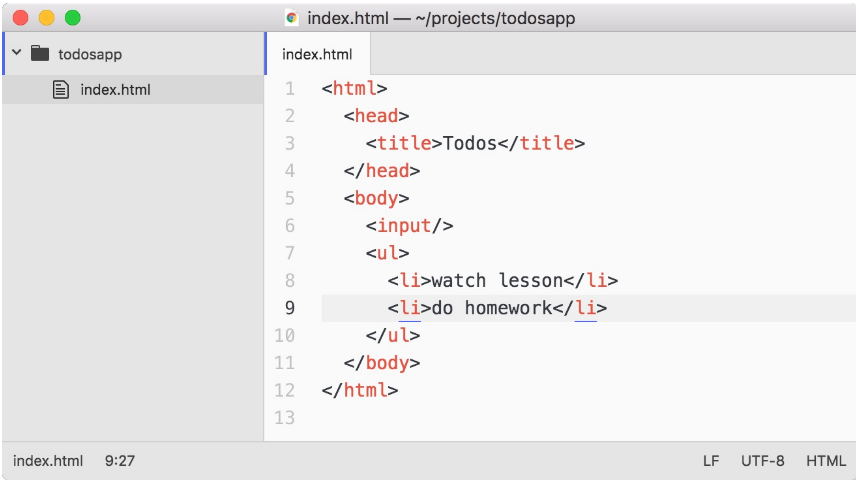
Perhaps you're thinking - why exactly "index"? Let it be one of your homework - reveal this secret by yourself... Google will help you;)

Code Editors

Before we continue, let's find out what is the code editor.

Any code is essentially a text that can be edited in a word processor. But the code has its own specifics in comparison with ordinary text. And as text editors have additional features such as tips on grammar, rulers, etc. - so editors specialized in editing code have its own special features to help in work with the code.

Here is the code of index.html opened in one of such code editors - Atom:



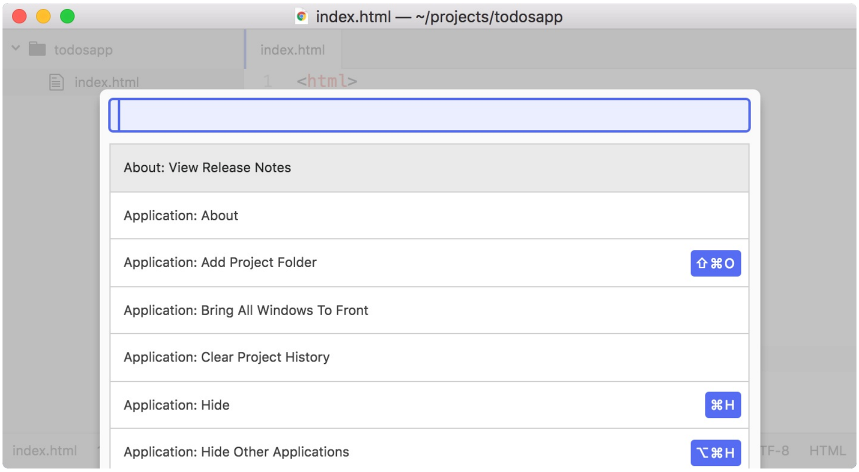
```html
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input/>
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```

index.html in the Atom editor

As you can see, reading of code is better because of the specific syntax highlighting. Now eyes are not blurred due to "plain text" mixed with "tagged elements".

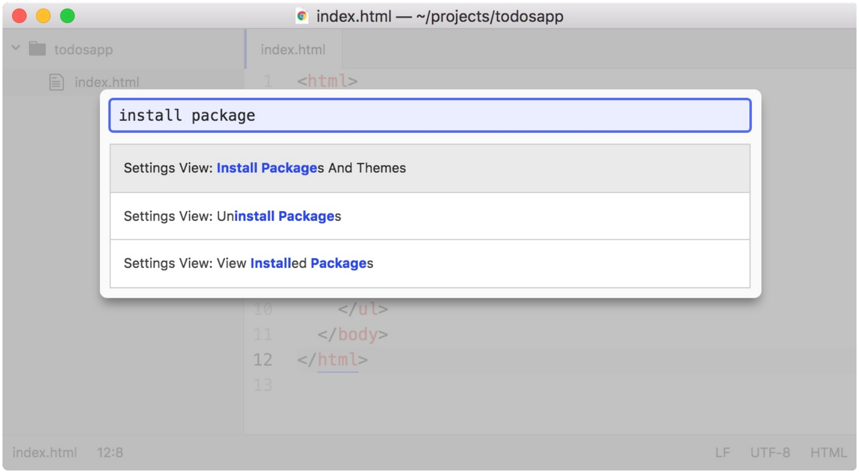
Another interesting thing about atom. There are many additional packages that can be installed and that can make our life easier by extending basic functionality of the editor with additional features. For example, it would be great to have a continuous "browser live view" of our code - the ability to enable browser emulator that reflect, in response to any change, how our code will look in the browser. Do we have such package for Atom? Don't know, let's search.

Cmd-shit-p for macs or *ctrl-shift-p* for windows will open the "Command Palette" dialog where we can search for and trigger any Atom feature.


index.html — ~/projects/todosapp
todosapp
index.html
index.html
<html>
About: View Release Notes
Application: About
Application: Add Project Folder ⇧⌘O
Application: Bring All Windows To Front
Application: Clear Project History
Application: Hide ⌘H
Application: Hide Other Applications ⌥⌘H
index.html
TF-8
HTML

Command Palette

Let's look for "install package".



```
install package
```

Settings View: Install Packages And Themes

Settings View: Uninstall Packages

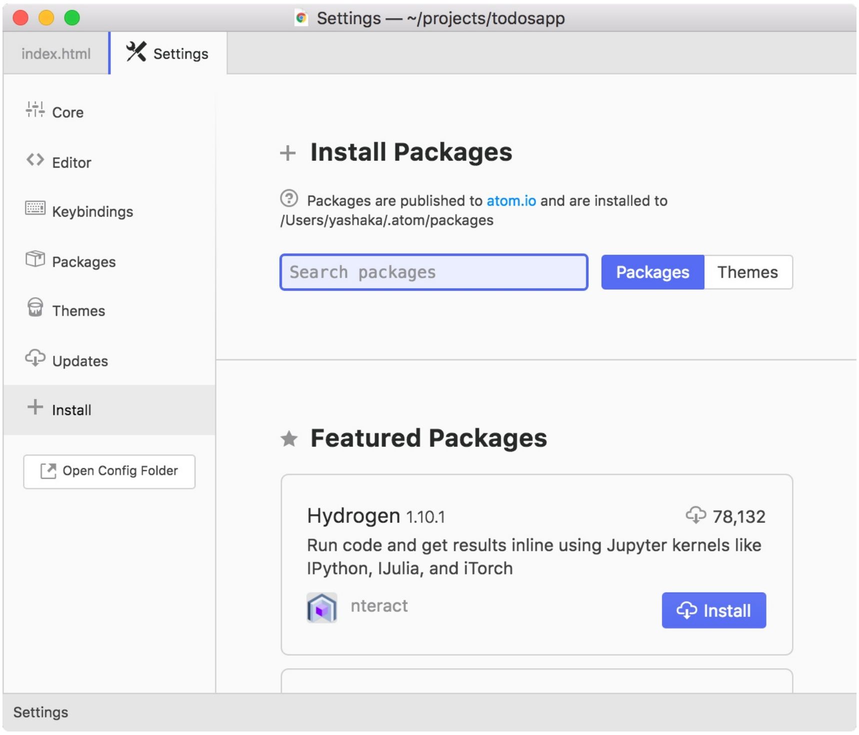
Settings View: View Installed Packages

```
<html>
      </ul>
    </body>
</html>
```

index.html 12:8

Search for "install package"

Great, seems like "Settings View: Install Packages And Themes" is the place where we can check available packages for Atom.

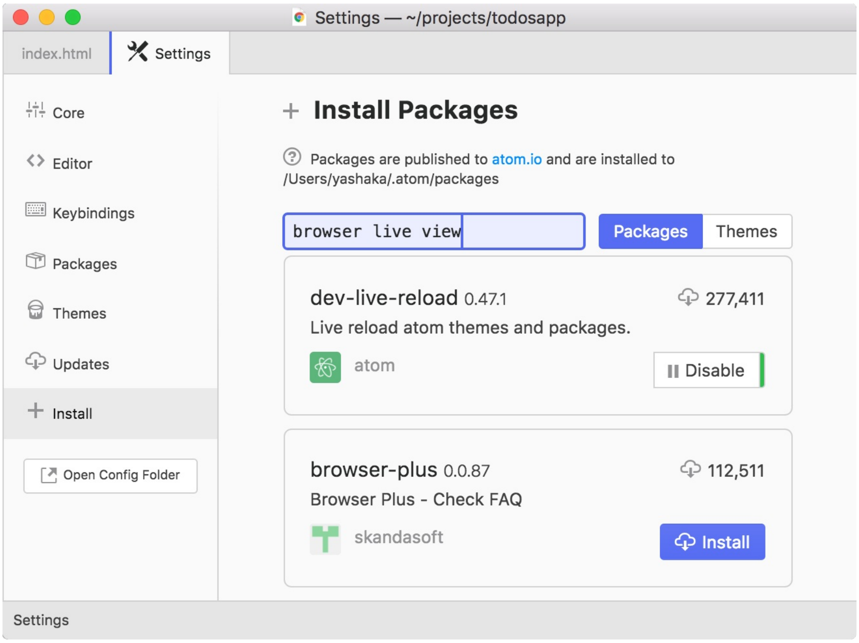


Packages are published to atom.io and are installed to /Users/yashaka/.atom/packages

Hydrogen 1.10.1 78,132
Run code and get results inline using Jupyter kernels like IPython, IJulia, and iTorch
nteract

Settings>Install

Now let’s go to "Search packages" and search for "browser live view".



Install Packages

Packages are published to atom.io and are installed to /Users/yashaka/.atom/packages

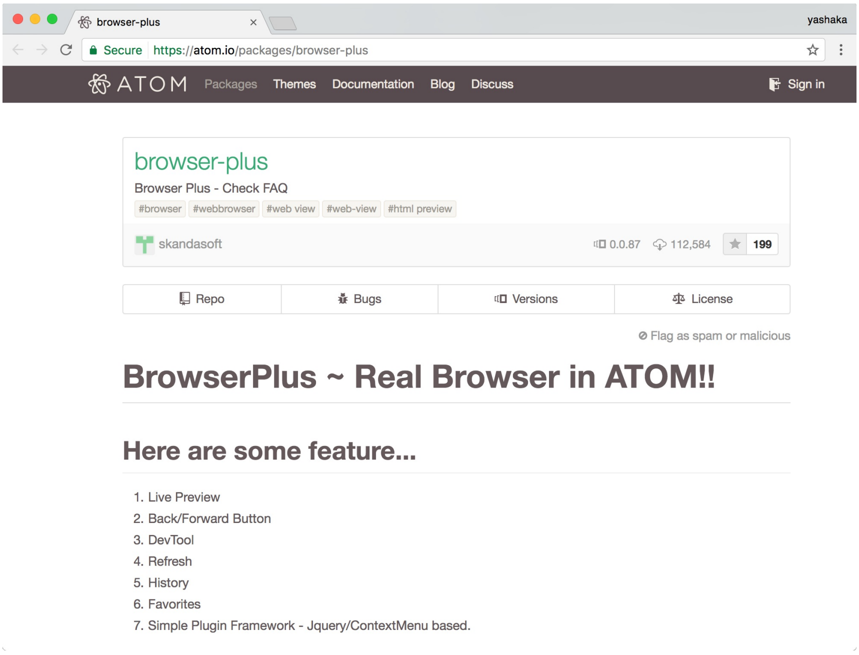
browser live view

dev-live-reload 0.47.1 — 277,411
Live reload atom themes and packages.
atom

browser-plus 0.0.87 — 112,511
Browser Plus - Check FAQ
skandasoft

Search for "browser live view"

Oh! Second item in the list is "browser-plus". If we click on its name, we get its public page with more details about the package:


browser-plus
yashaka
Secure https://atom.io/packages/browser-plus
ATOM Packages Themes Documentation Blog Discuss Sign in

browser-plus

Browser Plus - Check FAQ

#browser #webbrowser #web view #web-view #html preview

skandasoft 0.0.87 112,584 199

Repo Bugs Versions License

Flag as spam or malicious

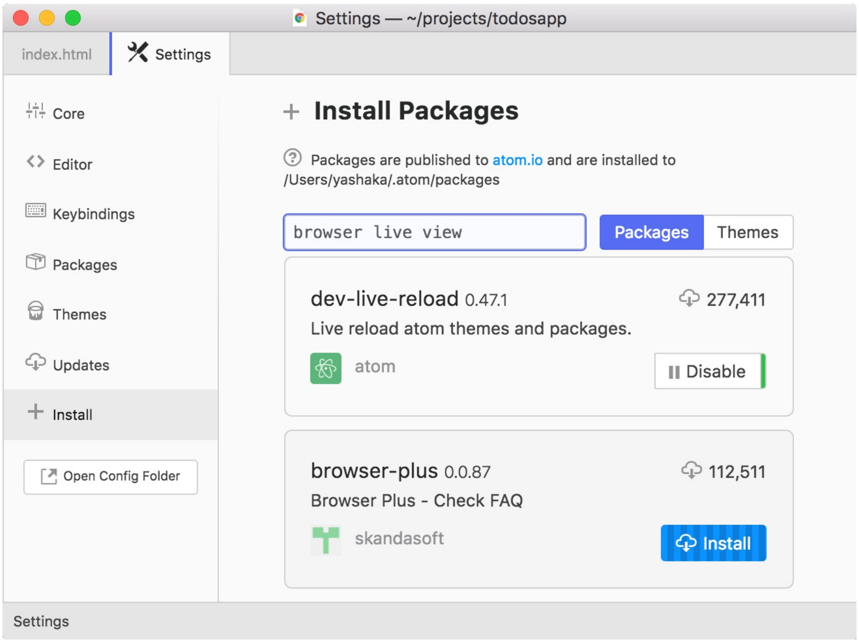
BrowserPlus ~ Real Browser in ATOM!!

Here are some feature...

1. Live Preview
2. Back/Forward Button
3. DevTool
4. Refresh
5. History
6. Favorites
7. Simple Plugin Framework - Jquery/ContextMenu based.

Browser-plus details

"Real Browser in ATOM"! And its first feature is "1. Live Preview". I think that it is what we need. Let's install this package.



+ Install Packages

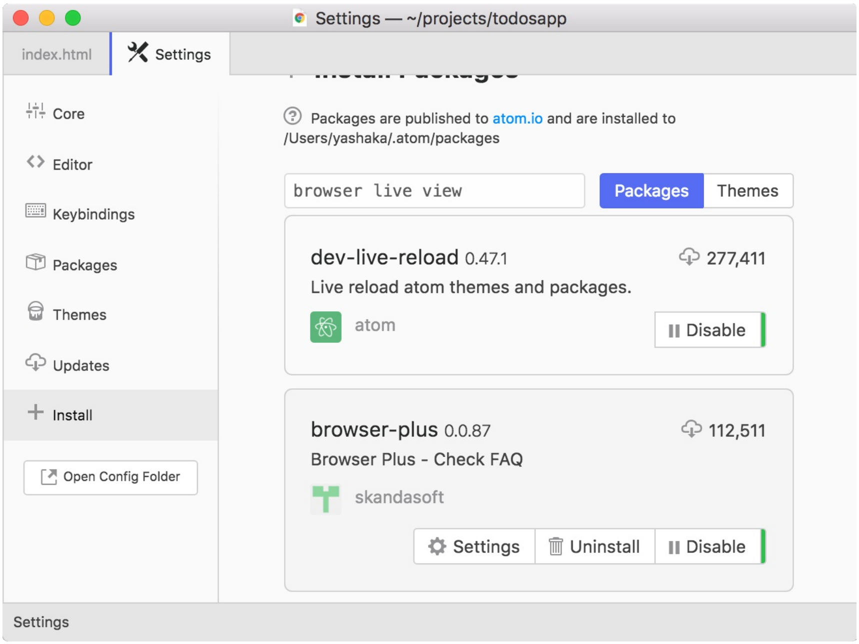
Packages are published to atom.io and are installed to /Users/yashaka/.atom/packages

browser live view

dev-live-reload 0.47.1 — 277,411
Live reload atom themes and packages.
atom

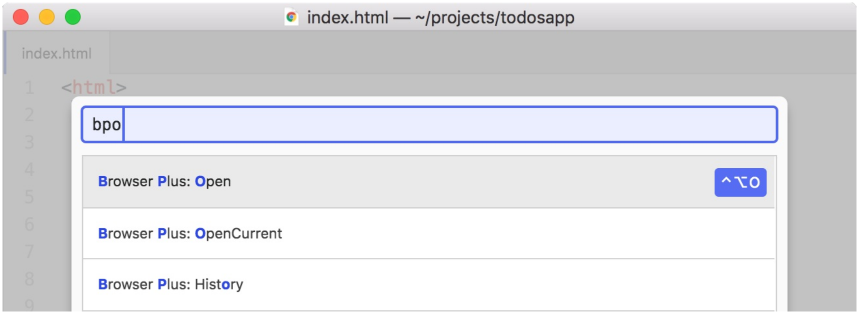
browser-plus 0.0.87 — 112,511
Browser Plus - Check FAQ
skandasoft

Installing "browser-plus" package

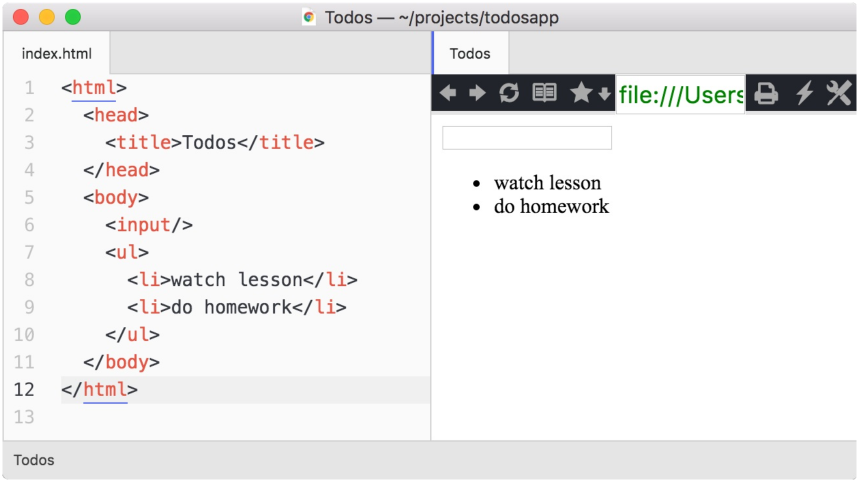


Installed "browser-plus" package

Now the "browser-plus" package is installed and we can toggle it using already known to us "Command Palette" (pay attention that we can search the needed command just by first letters of words from its name):



Toggling "browser plus"

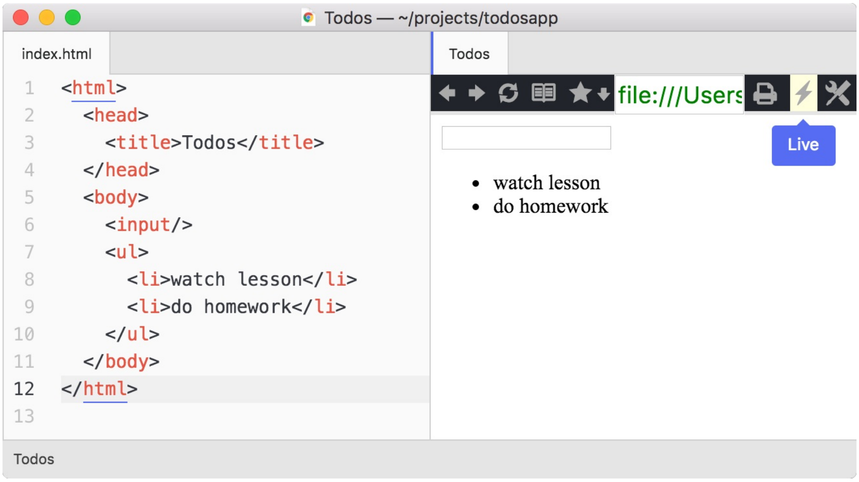


```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input/>
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```

- watch lesson
- do homework

Toggled "browser plus"

In order to enable the "live preview" feature, we need to press the corresponding "lightning" button:



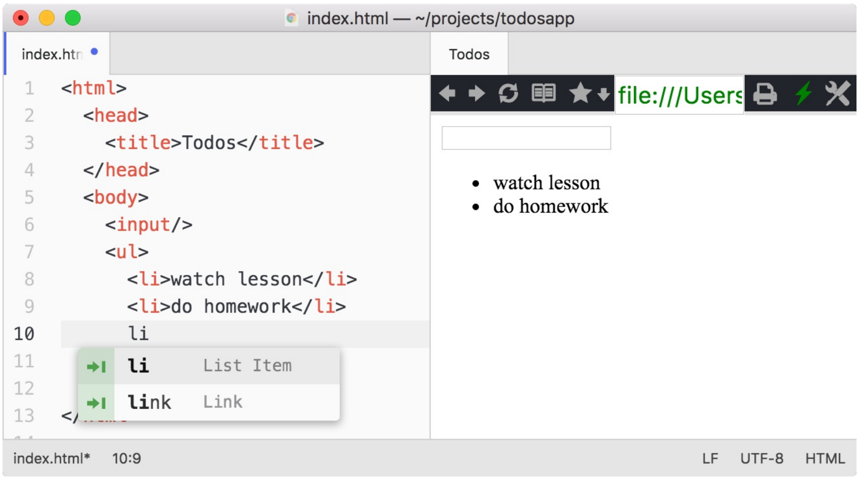
```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input/>
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```

- watch lesson
- do homework

Toggling "live preview"

Now once saved any new change to the file (via *cmd+s* on the mac or *ctrl+s* on the windows) the changes should be reflected in the browser emulator.

Let's, for example, add another task to the list. Here we can get familiar with one more feature of Atom - autocomplete. Let's start adding new `li` element just by entering `l` and `i` symbols (without tag symbols - `<` and `>`):



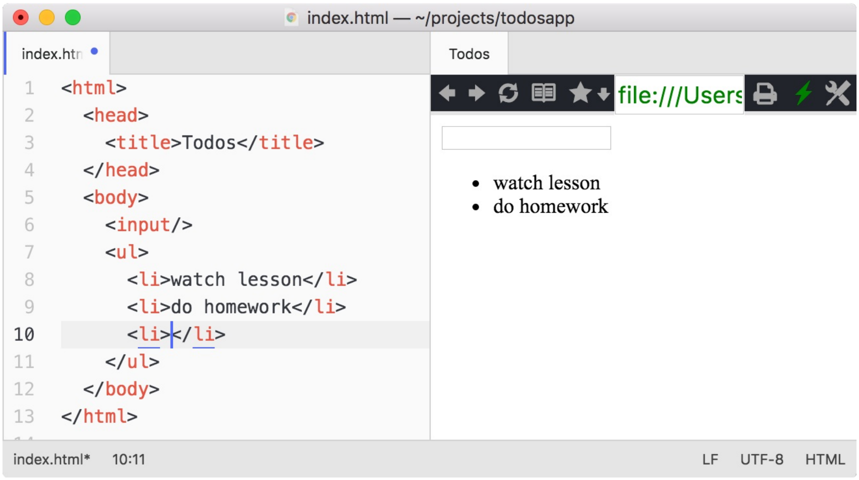
```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input/>
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
      li
```

```
li      List Item
link    Link
```

- watch lesson
- do homework

Trigger autocomplete hints

Now press `Tab` or `Enter` keyboard key to finalize "autocomplete":



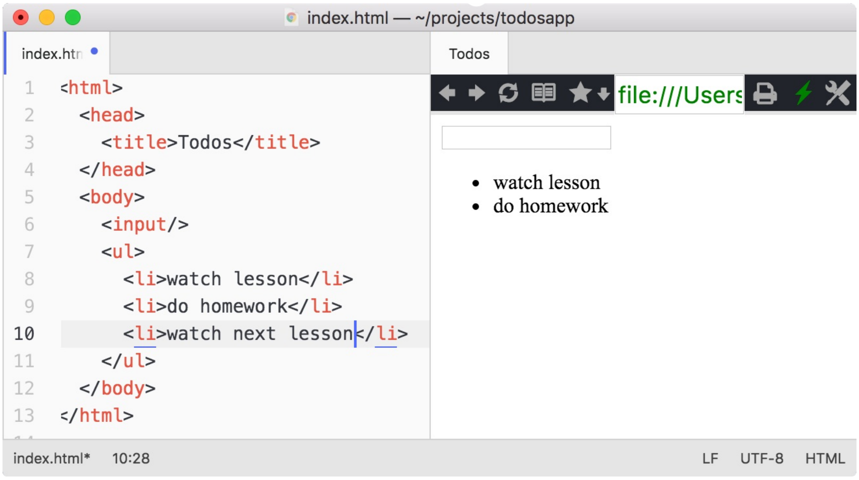
```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input/>
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
      <li></li>
    </ul>
  </body>
</html>
```

- watch lesson
- do homework

Finalize autocomplete

The editor will add opening tag's symbols (‹ and ›) and also will add the closing tag automatically.

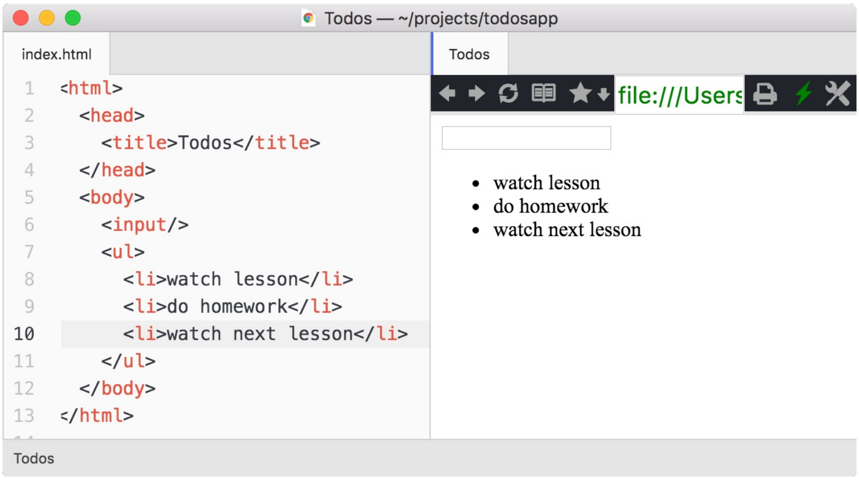
Now, on added text to the new task and hitted *cmd+s* on the Mac or *ctrl+s* on the Windows (saving changes) we will see them reflected in the browser emulator too:



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

- watch lesson
- do homework

Change before save



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

- watch lesson
- do homework
- watch next lesson

Reflected change after save

Awesome! isn’t it? :)

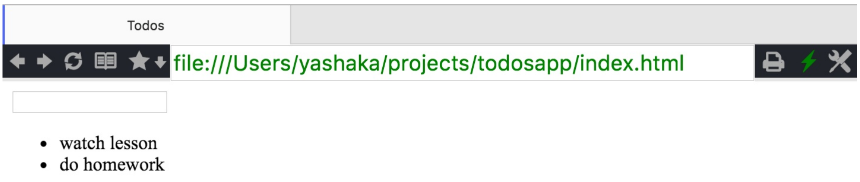
HTML Attributes

Specific HTML tags exist not for all known to us elements.

For example, most elements that involve "feedback from a user" - like entered text, selected checkbox or radiobutton, pressed button - they all can be implemented through the element with the tag `input` .

We have already used the `input` element previously to represent text field where user can enter text for a new task.

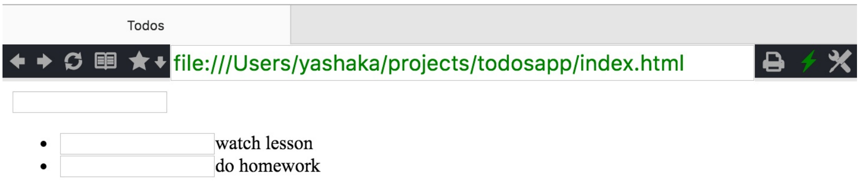
```
<html>
  <head>
    <title>Todos</title>
  </head>
  <body>
    <input />
    <ul>
      <li>watch lesson</li>
      <li>do homework</li>
    </ul>
  </body>
</html>
```



Default input as edit field

Just for example, let's now also add checkboxes for each task to give an ability to complete them or mark as "done":

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



- watch lesson
- do homework

More `input` elements to complete tasks

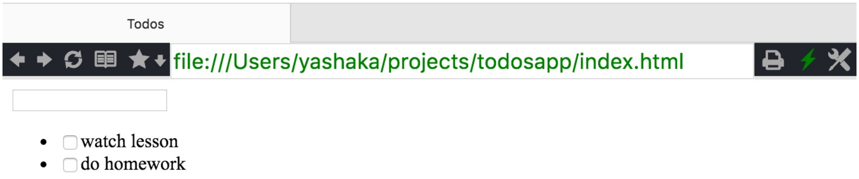
But how do we indicate now that "input" elements for tasks should be checkboxes?

In such situations when we need to provide more information about an element, or in other words - add some "identity" to our elements - then **html attributes** are used.

Attributes can be "general" that you can add to any element, and they can be "specific to certain elements", i.e. it makes sense to add them only to elements of certain tags. The latter regards, for example, the attribute `type` for the `input` element. In our case the `type` attribute allows us to precisely identify our `input` element as exactly checkbox:

```
<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>
```

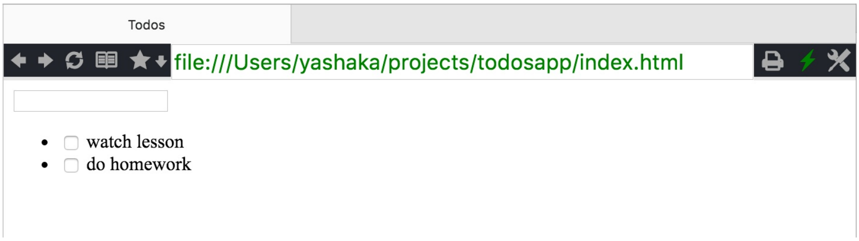


- ▢ watch lesson
- ▢ do homework

inputs as checkboxes

For better structure of our html markup and to identify more accurately each piece of task functionality - let's put our tasks' text into its own elements - labels:

```
<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>
```


Todos
file:///Users/yashaka/projects/todosapp/index.html

- ☐ watch lesson
- ☐ do homework

Labels

Now functionality "to complete a task" and functionality "to display the task text" are reflected in the markup by separate elements. In the future this will allow us to find the "needed functionality" in the code more conveniently and accurately. It is the same idea as to put things in order in the cupboard - each in its proper place.

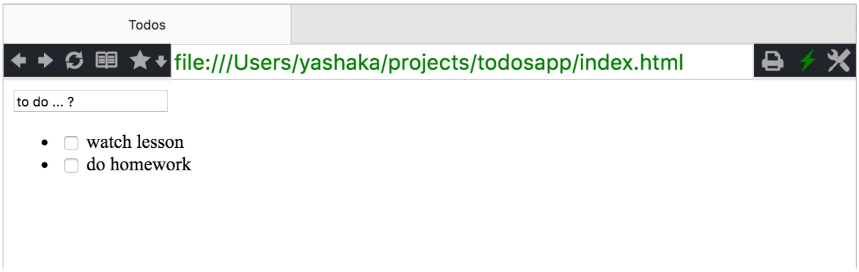
Let's now get familiar with some other attributes of "functional" type, which add important "features" to our elements.

How do you like the idea to add some text to the text field to serve as a hint for a user?

Here's how we can do this using the attribute `value` of the `input` element:

```
<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>
```



Input with value

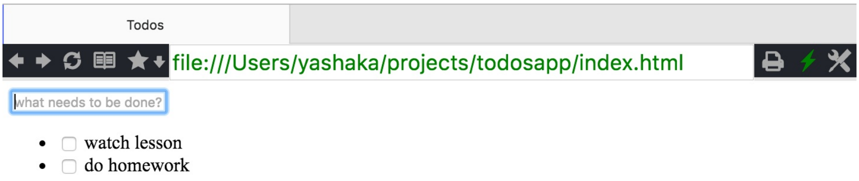
An "inverse relationship" works as well - if we enter a text into an edit box in the browser, then this text will be stored in the `value` attribute. Although, yet we do not know how to test this. But we'll return to this later — when we will use this feature in order to "spy" the value of `value` attribute of the text field after user pressed `Enter` , and then add a new task with the spied text to the list ;)

But wait, have written some text in the text field, now we are forcing the users to remove it each time before entering their own...

There is an easy way to fix this. It turns out that there is another attribute - `placeholder` - which is used exactly as a "user tip" that does not obstruct entering a new text:

```
<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>
```



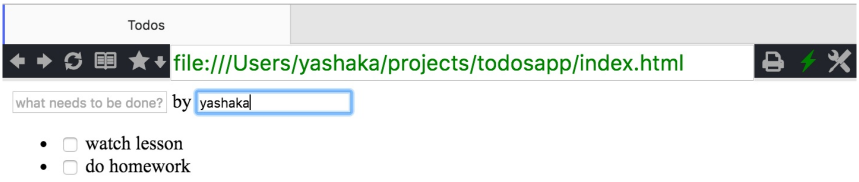
- watch lesson
- do homework

Input with placeholder

So, with help of attributes we can provide the Browser with additional information about some element. Browser is able to interpret a specific set of attributes and their values and is able to change how an element is displayed on the page correspondingly. For example, if the `input` element has the attribute `type="checkbox"` , then the Browser will display it as the checkbox. But not for all our wishes about style of element representation or some behaviour connected to it the predefined attributes with predefined set of valid values exist. For example, it would be useful change the style of the `input` element for the "entering new task text", so it is displayed in the center of the page. Even more useful would be to teach it to react to the pressing `Enter` key creating the new task with entered text earlier. For that we will have to write an additional code in additional files using another "languages". This code will find proper elements and change their style and behaviour. We will write this code later, and now let's think about the following. How such code can quickly find the proper input element among other input elements? How to distinguish the "text field" input element from the "checkbox" input element?

We could teach the program to seek *"element input that is not a checkbox"*, but what if we have another text field that displays the user's name who is assigned to this task?

```
<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>
```



Two text inputs

How to distinguish two text fields now? Not a bad idea would be to use search of the style - *"find the element input with attribute placeholder of the value What needs to be done".*

But what if our webpage supports 10 languages? Won't it be hard to list all options?

> *"Find the element input with attribute placeholder of the value 'What needs to be done?' or 'O que precisa ser feito?', or 'Що потрібно зробити?' or '需要做什麼呢' or ... "*

And we should notice that in the future our webpage can become more complicated and probably more new `input` elements will be added. All this will more complicate finding right element by our program "add a task by pressing `Enter` ".

To sum up - since we may have elements of the same type (i.e. with the same tag), but for different purposes, we need a clear way of marking elements to distinguish them.

There are special attributes that exist specifically for such purpose:

- Attribute `id` that is given to unique elements within a web page
- Attribute `class` that is given to elements belonging to a certain group

Thanks to such attributes we are able to tie needed functionality to relevant elements within the additional program, mentioned earlier.

So let's mark our `input` elements according to their roles:

```
<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>
```

Now we always have a clear way to distinguish one text field from another. And even collect a "sample" of checkboxes which are corresponded to class "toggle". This may be useful to give them specific unique style.

Exactly with tuning our web-application style we will be busy in next chapter;)

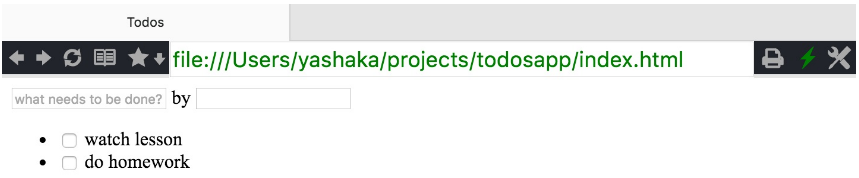
Adding unique style (CSS)

Adding custom styles to html page

HTML allows us to structure the content of the page - to mark up it's data reflecting their hierarchical nested structure.

```
<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>
```

The browser is able to visually represent this "structured data" with "default styles".



Browser renders default styles

Of course, it would be great to customize the style of visual representation of web pages data to our taste.

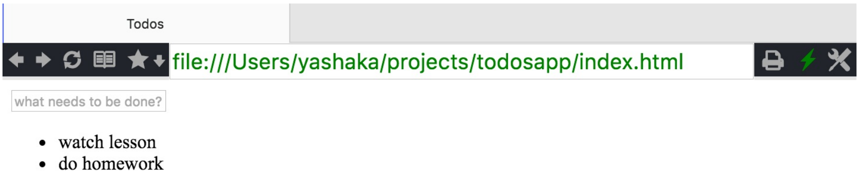
This can be achieved using another web development tool - **Cascading Style Sheets** or shortly **CSS**.

It is a special language that allows us to set **css rules** describing stylistic properties of the elements.

Usually styles are described in separate files with extensions `.css` .

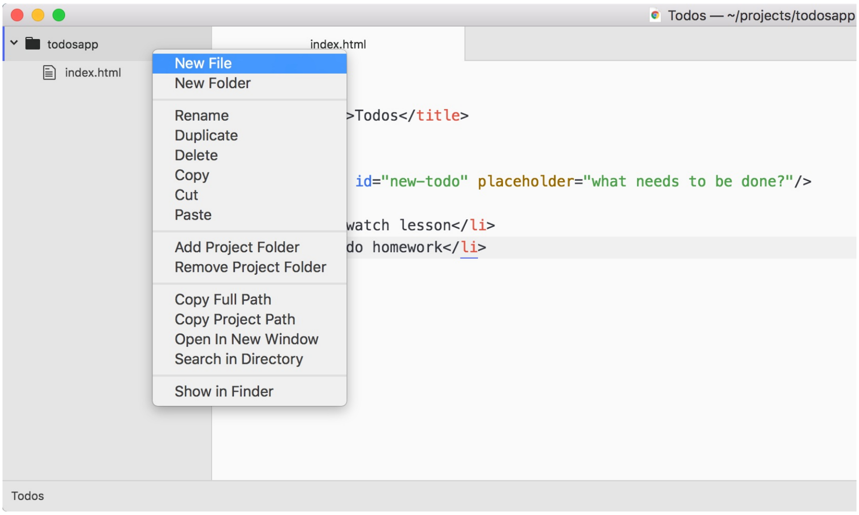
Before we write styles for our web application, let's simplify our HTML code... to the only structure that is actually needed to implement functionality of "adding tasks", not more. This will also make it easier to understand the overall process.

```
<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>
```



Simplified code in browser

Now, within our project (we can toggle projects' tree view by "cmnd+\" for mac or "ctrl+\" for windows) let's create a new file - `style.css`

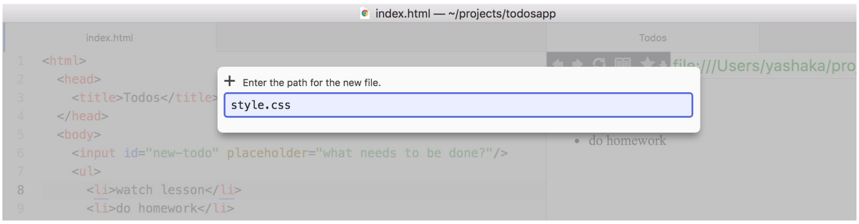


```
>Todos</title>

 id="new-todo" placeholder="what needs to be done?"/>

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

Creating new file through context menu from project's tree view


index.html — ~/projects/todosapp

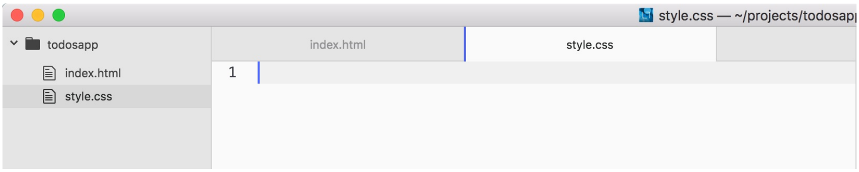
index.html

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

Enter the path for the new file.

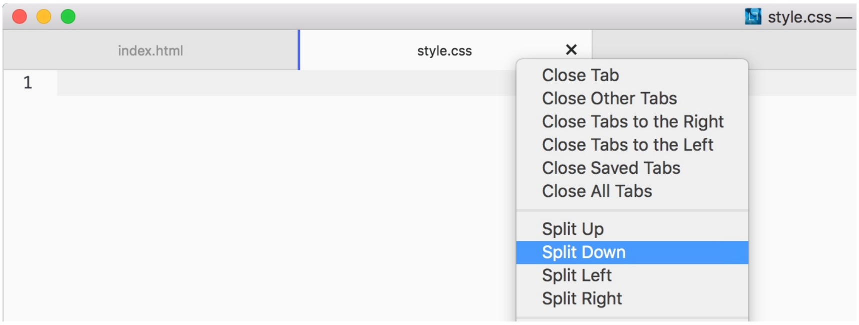
style.css

Setting new file name

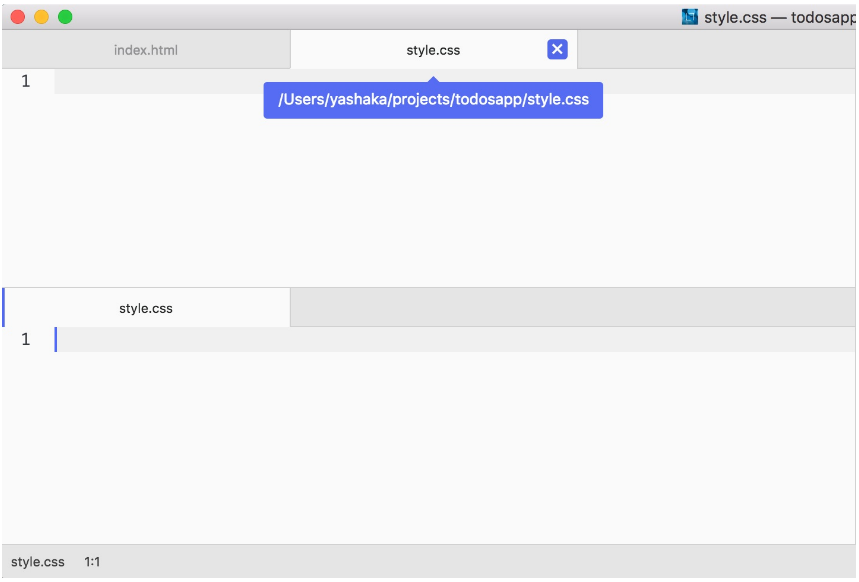


New file created

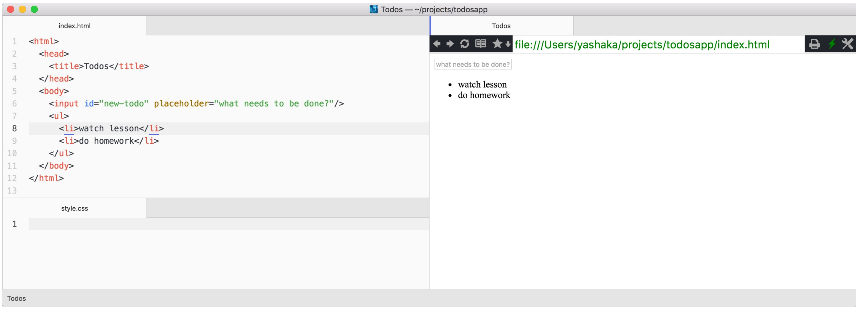
For convenience let's close project's tree view (by "cmnd+\" for mac or "ctrl+\" for windows) and move opened new file tab to a separate section within the code editor... by "splitting the window down":



closing the "duplicate":



and finally adjusting tabs-size accroding to our taste:


Todos — ~/projects/todosapp
index.html
style.css
Todos
file:///Users/yashaka/projects/todosapp/index.html
what needs to be done?
Todos

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

- watch lesson
- do homework

We have not written any "style rules" yet but let's just link our "css" file to html page, so our editor built-in browser can "apply" them continuously while writing and saving new code.

"Linking" is implemented via adding a special "register entry"

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

in "our page passport" - the `head` section:

```
<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>
```

CSS Rules

Well, let's start “decorating” our website :)

Let's start from a text field, don't you think it's too short?

Let's "stretch" it to the entire width of the page.

First, we are going to construct a **style rule** in an ordinary English language:

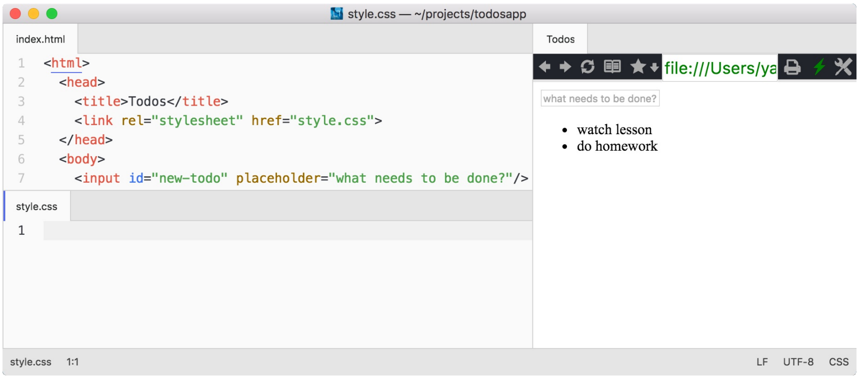
> *The element with `id="new-todo"` should have a width of all available space*

Or in more precise "technical" language:

> *The element with `id="new-todo"` must have the width of 100% of available space*

And here is the translation to the "CSS language":

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


style.css — ~/projects/todosapp

index.html

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

style.css

```

```

Todos

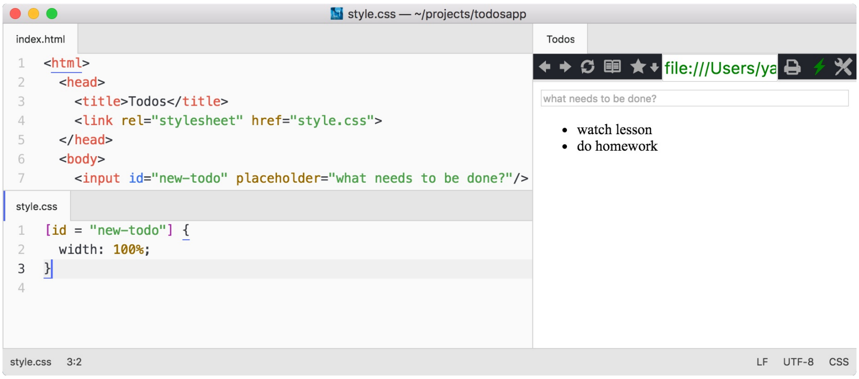
file:///Users/ya

what needs to be done?

- watch lesson
- do homework

style.css 1:1 LF UTF-8 CSS

Input with default length


style.css — ~/projects/todosapp

index.html

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

style.css

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

Todos

file:///Users/ya

what needs to be done?

- watch lesson
- do homework

style.css 3:2 LF UTF-8 CSS

Input with css 100% width

As we can see, the translation is quite clear:

The element with id="new-todo" ...

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

... must have ...

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

... the width of 100% of available space

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

CSS Selectors

The rule begins with a **selector** that determines the selection of elements to which the rule will be applied:

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

In this case - the elements that have attribute `id="new-todo"` (we have only one of such kind - especially for that we used the attribute for the unique identification of an element - `id`).

Square brackets syntax ...

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

is a universal way of saying - *"element or elements that have an attribute of the specific value"*. For example, we could "find our element" also through **selection by two attributes**:

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

Also, selector let us distinguish "search by attributes" from **"search by element tag"**:

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

Now our selector says:

> *"find the item(s)* ***with tag*** ***`input`*** *, with the attribute `id="new-todo"` , and attribute `placeholder="What needs to be done?"`*

In any case, because `id` is a special attribute that is unique for the element on the page - we can limit the search to be based only on selector **by `id` attribute**:

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

Moreover, it turns out that people so often search elements by the `id` attribute, that in CSS a shortcut exists:

> instead of

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

> we can write

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

CSS properties

Let's continue analysing the syntax of CSS-rules...

The element with `id="new-todo"` *must have the width of 100% of available space*

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

After the selector - in the curved brackets - follows the **block of rule definition**:

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

that lists "stylistic" **properties** that should be "set" for found element or elements:

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

Properties in the list are defined according to the following syntax:

property's name

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

colon

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

value

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

semicolon

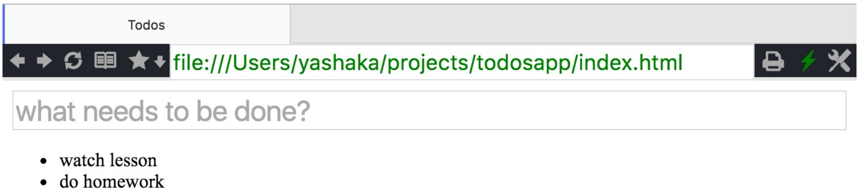
```
#new-todo {
     width: 100%;
}
```

More examples. Font properties

Let's add more properties to the list;)

Increase font size to 24 pixels:

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


Todos

file:///Users/yashaka/projects/todosapp/index.html

what needs to be done?

- watch lesson
- do homework

Make font italic:

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