# Learning Objectives

**Learners will be able to...**

- **Identify how HTML elements relate to each other in the DOM (Document Model Object) tree**
- **Write semantic, non-semantic, and style-based HTML elements:**
  - **Semantic - `html`, `header`, `section`, and `h1`**
  - **Non-semantic - `div` and `span`**
  - **Style-based - `strong`, `ol`, `ul`, `li`, `a`, `input`, `label`, `button` and comments**
- **Explain how the box model affects element positioning and layout on a webpage**
- **Write modular, semi-scalable CSS**

info

## Limitations

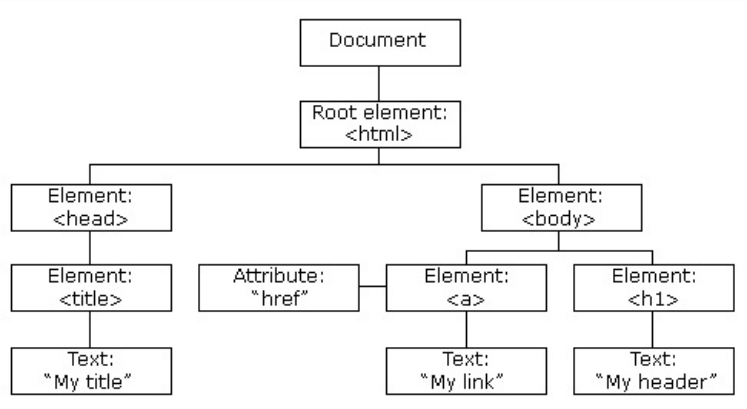
Only the most relevant HTML and CSS concepts that are associated with the todo app are covered in this assignment.

# HTML and the DOM

HTML is the standard markup language for Web pages. With HTML you can create your own website.

(source: https://www.w3schools.com/html/)

When a web page is loaded, the browser creates a Document Object Model of the page. The HTML DOM is constructed as a tree of Objects:




The HTML Document Object Model Object Tree

(source: https://www.w3schools.com/whatis/whatis_htmldom.asp)

This will be useful to know when we start building our todo app in VueJS. We'll revisit the HTML DOM later in the course, as we change items in our todo list from being in a "viewing" state to an "editing state" and check them off as "done."

For now, we'll review the basics of HTML, inserting it into the app skeleton that was made when we created our VueJS project.

## Creating `TodoApp.vue`

Delete `HelloWorld.vue` and instead, replace it with a component called `TodoApp.vue`. Add `TodoApp.vue` to the file tree in `src/assets/components`.

Next, rewrite `App.vue` so it looks like this:

```
<template>
  <TodoApp />
</template>

<script>
import TodoApp from './components/TodoApp.vue';

export default {
  name: 'App',
  components: {
    TodoApp,
  }
}
</script>

<style>
</style>
```

## Our Template Code

You probably already know that HTML is made up of tags and attributes. In VueJS and when we develop in MVC-flavored frameworks, these nested collections of tags that represent the HTML DOM are what we call **template code**. Templates hold the **view** of our application - what the user sees.

Copy and paste the following into `TodoApp.vue`.

```
<template>
  <div>
    <header>
        <h1>Todos</h1>
        <input autofocus type="text" placeholder="What needs to
        be done?">
    </header>
    <section>
        <input type="checkbox" checked="false">
        <label for="toggle-all">Mark all as complete</label>
        <ul>
            <li>
                <div>
                    <input type="checkbox">
                    <label>Todo Item</label>
                    <button>Remove Todo Item</button>
                </div>
                <!-- <input type="text" placeholder="Exiting
        todo to edit"> -->
```

```
            </li>
        </ul>
    </section>
    <footer>
        <span>
            <strong>1</strong>
            <span>item left</span>
        </span>
        <ul>
            <li>
                <a href="#/all">All</a>
            </li>
            <li>
                <a href="#/active">Active</a>
            </li>
            <li>
                <a href="#/completed">Completed</a>
            </li>
        </ul>
        <button>
            Clear completed todos
        </button>
    </footer>
  </div>
</template>

<script>
export default {
  // Your component definition goes here
}
</script>

<style scoped>
  /* Your CSS code goes here */
</style>
```

We'll fill out the content between the `<script>` tags in the next assignment.

## HTML is not case-sensitive

When you write HTML, your tags or elements and their attributes and respective values can be CAPITALIZED or lowercase. HTML is **case-insensitive**.

case-insensitive - treating or interpreting upper- and lowercase letters as being the same

(source: https://en.wiktionary.org/wiki/case_insensitive)

Curious to know more? Learn about the HTML standards here: https://www.w3.org/TR/2010/WD-html-markup-20101019/documents.html#syntax-document-html

## Doctypes

HTML is released in versions. Currently, we are on HTML5. Take a look at `index.html` in the `public` folder. At the top, you'll see a doctype declaration. All HTML documents must start with a `<!doctype>` declaration.

The `<!doctype>` declaration is not an HTML tag. It is *information to the browser* about what document type to expect.

Like the rest of HTML, the `<!doctype>` declaration is case-insensitive.

(source: https://www.w3schools.com/tags/tag_doctype.asp)

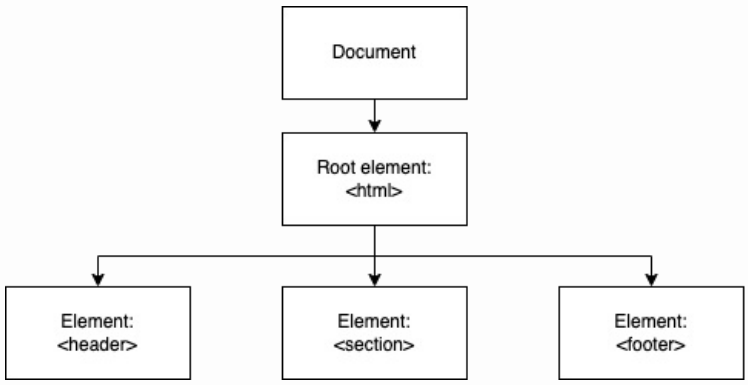
# HTML Headings, Lists, and Hyperlinks

The three main tags of our template are `<header>`, `<section>`, and `<footer>`. Notice here that we're doing two things:

1. We're adding indentations to make our HTML more easily readable.

You might return to this code in the future, long after you've forgotten it. This will make your own code more readable to your future self - and maybe other developers too!

We also add indentations to show the relationship between the child and parent HTML elements. This gives us an HTML DOM tree which looks like this:




.guides/img/dom-tree

2. These tags - `<header>`, `<section>`, and `<footer>` - are called **semantic elements**.

A semantic element clearly describes its meaning to both the browser and the developer.

A semantic tag encloses and explicitly defines its content. That is, a `<form>` tag contains a form to be submitted. A `<table>` tag contains data organized in a table. And an `<article>` tag will contain independent, self-contained content, such as a forum post, blog post, or news story with an author, timestamp, title, and post body and elements for each entity.

(source: https://www.w3schools.com/html/html5_semantic_elements.asp)
(source: https://www.w3schools.com/tags/tag_article.asp)

## Heading tags

Under the `<header>`, we've added a heading tag. In HTML5, there are six heading tags:

- `h1`
- `h2`
- `h3`
- `h4`
- `h5`
- `h6`

*Semantically*, `<h1>` defines the most important heading. `<h6>` defines the least important heading.

(source: https://www.w3schools.com/tags/tag_hn.asp)

`<h1>` is being used at the top of our todo list web app and is the most important heading.

## Comments

We're going to create a fully-functional todo list web app, but first let's add two things: comments for our future work on the app and `<div>` containers for our content.

A comment in HTML is a unique tag that encases text which does not display on your webpage. You can add comments to your HTML with the `<!-- ... -->` tag, enclosing text between the opening and closing angle brackets.

Why do we include comments in our code? It's for you as a developer to document what you need to document in the code itself - a useful thing if you revisit your code in the future and don't remember exactly what it contains.

(source: https://www.w3schools.com/tags/tag_comment.asp)

```
<section>
    <header>
        <h1>Todos</h1>
        <!-- TODO: Add input field so we can add todo items to
        our list. -->
    </header>
    <section>
        <!-- TODO: Add checkbox to mark all as completed. -->
        <div>
            <!-- TODO: Iterate over todo items and list them
        out. -->
            <!-- TODO: Mark as completed, view, or delete todo
        item -->
        </div>
        <!-- TODO: Edit todo item -->
    </section>
    <footer>
        <span>
            <strong>1</strong>
            <span>item left</span>
        </span>
        <!-- TODO: Add filters: all, active, completed. -->
        <!-- TODO: Add button to clear completed todo items. -->
    </footer>
</section>
```

## Non-semantic elements

Some examples of commonly used non-semantic elements include `<div>` and `<span>` for ***divisions*** of elements and ***spans*** of inline content, respectively. Unlike semantic elements, non-semantic elements don't give us a clear idea as to what type of content is within them. A `<div>` can divide a page or it can just be a container for any type of content. Likewise, a `<span>` can contain text, take the form of a button, hold a link to another page in it.

Although these tags are abbreviated and have clear definitions, we can't assume what's contained between their opening and closing elements.

## Block and inline elements

A `<div>` is also known as a **block-level element**.

(source: https://www.w3schools.com/tags/tag_div.asp)

A block-level element has several properties:
1. it always starts on a new line on the webpage
2. it automatically has upper and lower spacing (margin) added to it
3. it always takes up the full width available within its parent element, stretching out as far left and right as it can
4. it always occupies vertical space equal to the height of its contents, creating a "block"

*Block-level elements* will come up again when we learn about Cascading Style Sheets (CSS) in section 2 of this module.

A `<span>` tag is a lot like the `<div>` tag, except it's an inline-level element.

Unlike a block-level element, an inline element:
1. does not start on a new line
2. only takes up as much width as necessary

(source: https://www.w3schools.com/tags/tag_span.asp)
(source: https://www.w3schools.com/html/html_blocks.asp)

## Bold or `<strong>` text

A `<strong>` tag defines important text and its default behavior is to make text appear **bold**.

(source: https://www.w3schools.com/tags/tag_strong.asp)

We've removed the comment `<!-- TODO: List number of remaining todo items. -->` and added some content as a placeholder - a `<strong>` and a `<span>`.

```
<section>
    <header>
        <h1>Todos</h1>
        <!-- TODO: Add input field so we can add todo items to
        our list. -->
    </header>
    <section>
        <!-- TODO: Add checkbox to mark all as completed. -->
        <div>
            <!-- TODO: Iterate over todo items and list them
        out. -->
            <!-- TODO: Mark as completed, view, or delete todo
        item -->
        </div>
        <!-- TODO: Edit todo item -->
    </section>
    <footer>
        <span>
            <strong>1</strong>
            <span>item left</span>
        </span>
        <!-- TODO: Add filters: all, active, completed. -->
        <!-- TODO: Add button to clear completed todo items. -->
    </footer>
</section>
```

## Lists and list items - `<ol>`, `<ul>`, `<li>`

In HTML, we can have ordered, unordered, and description lists.

An ordered list `<ol>` is a number-ordered list.
An unordered list `<ul>` appears as a series of bullet points.
A description list `<dl>` can be used with words and their respective definitions.

Ordered and unordered have list items which use the `<li>` tag.
A description list has two child tags, a description tag `<dt>` and and a description definition tag `<dd>`

(source: https://www.w3schools.com/html/html_lists.asp)

For our todo list, we're going to use ordered and unordered lists, replacing the comments:

and

with an ordered list tag and just one list item (for now!):

```
<ol>
    <li>
        <div>
            <!-- TODO: Mark as completed, view, or delete todo
        item -->
        </div>
        <!-- TODO: Edit todo item -->
    </li>
</ol>
```

Next, we'll define filters for our all, active, and completed todo items:

```
<ul>
    <li>
        All
    </li>
    <li>
        Active
    </li>
    <li>
        Completed
    </li>
</ul>
```

And with some appropriate CSS styling, this will look like a series of inline items instead of a series of bullet points.

Putting these completed lists into our page, we get:

```
<section>
    <header>
        <h1>Todos</h1>
        <!-- TODO: Add input field so we can add todo items to
        our list. -->
    </header>
    <section>
        <!-- TODO: Add checkbox to mark all as completed. -->
        <ol>
            <li>
                <div>
                    <!-- TODO: Mark as completed, view, or
        delete todo item -->
                </div>
                <!-- TODO: Edit todo item -->
            </li>
        </ol>
    </section>
    <footer>
        <span>
            <strong>1</strong>
            <span>item left</span>
        </span>
        <ul>
            <li>
                All
            </li>
            <li>
                Active
            </li>
            <li>
                Completed
            </li>
        </ul>
        <!-- TODO: Add button to clear completed todo items. -->
    </footer>
</section>
```

## Hyperlinks - <a>

Hyperlinks connect HTML pages together. The word “hyper” in this context doesn’t refer to an emotional state, but rather the ability for pages to extend. HyperText is **so** important in HTML that it’s in the name: HyperText Markup Language!

(source: https://en.wikipedia.org/wiki/Hypertext)

The `<a>` tag defines a hyperlink, linking one page to another. When we get into VueJS, we'll define a `method` in our `Single-File Component (SFC)` to change the todo items that are displayed when a user clicks a filter link.

For now, though, we'll change our filter items to links by simply adding an `<a>` tag and an `href attribute`.

attribute - provide more information about HTML elements with a key/value pair

(source: https://www.w3schools.com/html/html_attributes.asp)
(source: https://www.w3schools.com/tags/ref_attributes.asp)

href - defines the URL of the page a link goes to

(source: https://www.w3schools.com/tags/att_href.asp)

Turning our filter for all todo items into a link requires enclosing text in an `<a>` tag and adding the `href` attribute with its respective value.

The # symbol here points our web browser to a specific spot on our webpage - in this case, the filter `method` that we will add to our VueJS `Single-File Component (SFC)`.

Let's turn our filter for all todo items into a hyperlink.

```
<a href="#/all">All</a>
```

```
<section>
    <header>
        <h1>Todos</h1>
        <!-- TODO: Add input field so we can add todo items to
        our list. -->
    </header>
    <section>
        <!-- TODO: Add checkbox to mark all as completed. -->
        <ul>
            <li>
                <div>
                    <!-- TODO: Mark as completed, view, or
        delete todo item -->
                </div>
                <!-- TODO: Edit todo item -->
            </li>
        </ul>
    </section>
    <footer>
        <span>
            <strong>1</strong>
            <span>item left</span>
        </span>
        <ul>
            <li>
                <a href="#/all">All</a>
            </li>
            <li>
                <a href="#/active">Active</a>
            </li>
            <li>
                <a href="#/completed">Completed</a>
            </li>
        </ul>
        <!-- TODO: Add button to clear completed todo items. -->
    </footer>
</section>
```

# HTML Form Fields and Elements

## Form fields and elements - `<label>`, `<button>`, `<input>`

An HTML form is used to collect user input and information. When we get started with VueJS, we'll be able to create, edit, and delete todo items from our app using `methods` in our `Single-File Component (SFC)`.

The appearance of an `<input>` can change depending on the value of its `type` attribute. An `<input>` tag specifies a field where a user can enter data.

(source: https://www.w3schools.com/tags/tag_input.asp)

The `<label>` tag will define the label for our `<input>` form fields. The `for` attribute connects the label to its respective form field, as long as that form field element has the exact same value for its `id` attribute.

(source: https://www.w3schools.com/tags/tag_label.asp)

A `<button>` can take the form of text or even an image. In our VueJS application, we'll connect our `<button>` to a method which clears completed todo items.

(source: https://www.w3schools.com/tags/tag_button.asp)

## Adding `<input>` fields

We'll now replace the remaining comments in our web app with code. Let's begin by replacing this comment:

```
<!-- TODO: Add input field so we can add todo items to our list.
        -->
```

with this:

```
<input autofocus type="text" placeholder="What needs to be
        done?">
```

We're applying the `autofocus` attribute so that when the page loads, this `<input>` automatically has a keyboard cursor in it, allowing you to add todo items to your list.

The `placeholder` attribute specifies a short hint or reminder to you about what kind of data is expected in your `<input>`.

(source: https://www.w3schools.com/tags/tag_input.asp)

## Checkboxes

For our next comment to be replaced:

```
<!-- TODO: Add checkbox to mark all as completed. -->
```

We're also going to add a `<label>` to clearly communicate what the checkbox does.

```
<input id="toggle-all" type="checkbox" checked="false">
<label for="toggle-all">Mark all as complete</label>
```

## Elements to interact with our todo items

We'll have three elements replace the comment below:
- something to mark items as completed
- a form field to just view todo items
- a way to delete todo items

```
<!-- TODO: Mark as completed, view, or delete todo item -->
```

So let's make:
- a checkbox to mark todo items as completed
- a label enclosing a single todo item
- a button to remove todo items from our list

```
<input type="checkbox">
<label>Todo Item</label>
<button>Remove Todo Item</button>
```

## An editable `<input>` field

In the VueJS lesson, we're going to attach a method to a todo item so that when it's clicked, it goes into edit mode. For now, this comment:

```
<!-- TODO: Edit todo item -->
```

is going to be replaced by an input field of `type="text"` and a `placeholder` that hints to the user that this is an editing field.

```
<!-- <input type="text" placeholder="Exiting todo to edit"> -->
```

For now, we're going to comment this `<input>` out and we'll make it reappear in our VueJS lesson.

## Adding a `<button>` to clear completed todo items

Lastly, there's this comment:

```
<!-- TODO: Add button to clear completed todo items. -->
```

which will be replaced with a simple button with text "Clear completed todos".

```
<button>
    Clear completed todos
</button>
```

The final skeleton for our todo app (which was what you were told to copy into your code file initially) looks like this:

```
<template>
  <div>
    <header>
        <h1>Todos</h1>
        <input autofocus type="text" placeholder="What needs to
        be done?">
    </header>
    <section>
        <input type="checkbox" checked="false">
        <label for="toggle-all">Mark all as complete</label>
        <ul>
            <li>
                <div>
                    <input type="checkbox">
                    <label>Todo Item</label>
                    <button>Remove Todo Item</button>
                </div>
                <!-- <input type="text" placeholder="Exiting
        todo to edit"> -->
            </li>
```

```
        </ul>
    </section>
    <footer>
        <span>
            <strong>1</strong>
            <span>item left</span>
        </span>
        <ul>
            <li>
                <a href="#/all">All</a>
            </li>
            <li>
                <a href="#/active">Active</a>
            </li>
            <li>
                <a href="#/completed">Completed</a>
            </li>
        </ul>
        <button>
            Clear completed todos
        </button>
    </footer>
  </div>
</template>

<script>
export default {
  // Your component definition goes here
}
</script>

<style scoped>
  /* Your CSS code goes here */
</style>
```

Then follow the directions below to execute your application.

create

## Executing Your VueJS Application

1. Navigate to the `todomvc` directory by entering `cd todomvc` in the Terminal.
2. Execute the application by entering `npm run serve` in the Terminal.
3. Navigate back to the initial browser tab.
4. Finally, click on the reload button to display your application.

## Exiting Your VueJS Application

While your VueJS application is currently running, press the following keyboard key combinations to exit the application: `control` + `c` or `ctrl` + `c`

If successful, your app should look like this:


.guides/img/no-styles

But it looks a little plain ...

# CSS

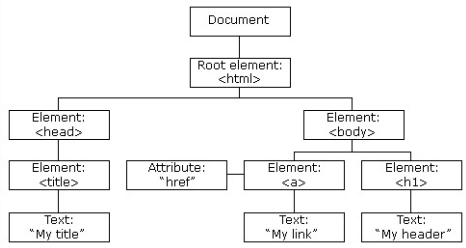
## Style - What is CSS?

CSS stands for Cascading Style Sheets. It is the language we use to style a webpage, describing how HTML elements are to be displayed on screen, paper, or in other media.

(source: https://www.w3schools.com/css/css_intro.asp)

## The Box Model

Recall that in HTML, each element part of the Document Object Model (DOM) Tree, a tree-like representation of the elements on our webpage.




The HTML Document Object Model Object Tree

In addition, each element is also visually surrounded by a box, allowing us to decorate its:
- margin
- border
- padding
- content

These four items - margin, border, padding, and content - make up the box model.

The Box Model

Margin is outside of an element's border, creating space between elements on a page.

The border of an element goes around an element, separating an element's padding and margin.

Padding is the space between the content and border of an element.

And the contentis what is between our opening and closing HTML element tags - text, images, and more HTML elements.

(source: https://www.w3schools.com/css/css_boxmodel.asp)

```
<style>
    div {
        background-color: green;
        border: 5px solid red;
        margin: 5px 10px 15px 20px;
        padding: 20px 15px 10px 5px;
    }
</style>

<html>
    <body>
        <div>
            Here is some content.
        </div>
    </body>
</html>
```

Change the colors of the `background-color` and `border` properties.

You can see that both the `margin` and `padding` CSS attributes have four values, denoted in pixel values. In order, these values indicate the top, right, bottom, and left sides of the margin and padding of our `<div>` element.

You can easily remember this with the mnemonic device: **Going clockwise, Margin and Padding are TRouBLe!**

## Attributes for styling elements: `class` and `id`

Earlier in the section, we learned about `attributes`.

attribute - provide more information about HTML elements with a key/value pair

In the last section, we applied styles to the `<div>` element.

We can also apply CSS styling to our webpage by defining styles using the `class` and `id` HTML attributes. Classes are accessed using the period (`.`) and IDs are accessed using the pound or hash (`#`) symbol.

```
<style>
    div {
        background-color: green;
        border: 5px solid red;
        margin: 5px 10px 15px 20px;
        padding: 20px 15px 10px 5px;
    }

    .content-box {
        border: 1px solid black;
    }

    #first {
        background-color: blue;
    }

    #second {
        background-color: red;
        border: 5px solid black;
    }

    #third {
        background-color: purple;
    }
</style>

<html>
    <body>
        <div class="content-box" id="first">
            Here is some content.
        </div>

        <div class="content-box" id="second">
            Here is some content.
        </div>

        <div class="content-box" id="third">
            Here is some content.
        </div>
    </body>
</html>
```

(source: https://www.w3schools.com/cssref/sel_class.php)
(source: https://www.w3schools.com/cssref/sel_id.php)

## CSS Specificity

Notice how even though we have the style `background-color: 1px solid black;` applied to our `<div>` elements with the `content-box` class, the `<div>` element with `id="second"` has a thicker border applied to it. Why does this happen?

Specificity is a scoring system in CSS that styles elements according to their calculated rank.

In this system, each style has a score:
- inline styles (for example, `<div style="border: 1px solid black;">`) have a score of **1000**
- IDs have a score of **100**
- Classes have a score of **10**
- Elements (just a `<div>` with no `style`, `class`, or `id` attribute) have a score of **1**)

So when we apply a style:

```
#second {
    background-color: red;
    border: 5px solid black;
}
```

and it has a score of 100, it overrides the style

```
.content-box {
    border: 1px solid black;
}
```

which has a score of 10

and also the `background-color` applied to our `<div>` element

```
div {
    background-color: green;
    border: 5px solid red;
    margin: 5px 10px 15px 20px;
    padding: 20px 15px 10px 5px;
}
```

which just has a score of 1.

```
<style>
    div {
        background-color: green;
        border: 5px solid red;
        margin: 5px 10px 15px 20px;
        padding: 20px 15px 10px 5px;
    }

    .content-box {
        border: 1px solid black;
    }

    #first {
        background-color: blue;
    }

    #second {
        background-color: red;
        border: 5px solid black;
    }

    #third {
        background-color: purple;
    }
</style>

<html>
    <body>
        <div class="content-box" id="first">
            Here is some content.
        </div>

        <div class="content-box" id="second">
            Here is some content.
        </div>

        <div class="content-box" id="third">
            Here is some content.
        </div>
    </body>
</html>
```

## Adding classes and styles to our todo list app

Now that we've learned a little bit about CSS, let's decorate our todo list app with styles!

Copy this code into `App.vue` and paste it between the `<style>` tags.

```
<style>
/* body */
body {
    font: 14px 'Helvetica Neue', Helvetica, Arial, sans-serif;
    line-height: 1.4em;
    background: #f5f5f5;
    color: #111111;
    min-width: 230px;
    max-width: 550px;
    margin: 0 auto;
}

/* button */
button {
    border: none;
    background: none;
}

/* .todoapp */
.todoapp {
    background: #fff;
    margin: 130px 0 40px 0;
    position: relative;
}
```

```css
/* h1 */
.todoapp h1 {
    position: absolute;
    top: -140px;
    width: 100%;
    font-size: 80px;
    font-weight: 200;
    text-align: center;
    color: #b83f45;
}

/* placeholder */
.todoapp input::placeholder {
    font-style: italic;
    color: rgb(0, 0, 0);
}

/* input::new/edit */
.new-todo,
.edit {
    width: 100%;
    font-size: 24px;
    box-sizing: border-box;
}

.new-todo {
    padding: 16px 16px 16px 60px;
    border: none;
}

/* main, toggle-all */
/* relative aligns the new-todo input and down-facing chevron */
.main {
    position: relative;
}

.toggle-all {
    opacity: 0;
    position: absolute;
}

.toggle-all + label {
    display: flex;
    align-items: center;
    justify-content: center;
    width: 45px;
    height: 65px;
    font-size: 0;
    position: absolute;
    top: -65px;
}

.toggle-all + label:before {
    content: '❯';
    font-size: 22px;
    color: #949494;
    padding: 10px 27px 10px 27px;
    transform: rotate(90deg);
  cursor: pointer;
}

.toggle-all:checked + label:before {
    color: #484848;
}

/* todo-list */
.todo-list {
    margin: 0;
    padding: 0;
    list-style: none;
}

/* todo list items and states */
.todo-list li {
    position: relative;
    font-size: 24px;
    border-bottom: 1px solid #ededed;
```

```
}

.todo-list li:last-child {
    border-bottom: none;
}

.todo-list li.editing {
    border-bottom: none;
    padding: 0;
}

/* display: block - otherwise it's hidden */
/* margin aligns it under the previous input */
.todo-list li.editing .edit {
    display: block;
    width: calc(100% - 43px);
    padding: 12px 16px;
    margin-left: 43px;
}

.todo-list li label {
    word-break: break-all;
    padding: 15px 15px 15px 60px;
    display: block;
    color: #484848;
}

.todo-list li.completed label {
    color: #949494;
    text-decoration: line-through;
}

.todo-list li .edit {
    display: none;
}

.todo-list li.editing .view {
    display: none;
}

.todo-list li .destroy {
    display: none;
    position: absolute;
  cursor: pointer;
    top: 5px;
    right: 10px;
    width: 40px;
    height: 40px;
    font-size: 30px;
    color: #949494;
}

.todo-list li .destroy:hover,
.todo-list li .destroy:focus {
    color: #C18585;
}

.todo-list li .destroy:after {
    content: '×';
}

.todo-list li:hover .destroy {
    display: block;
}

/* edit which hides checkbox */
.todo-list li .toggle {
    width: 40px;
    height: 40px;
    position: absolute;
    margin: auto 0;
    opacity: 0;
  cursor: pointer;
}

/* circle, no check? */
/*
```

```
    Firefox requires `#` to be escaped -
        https://bugzilla.mozilla.org/show_bug.cgi?id=922433
    IE and Edge requires *everything* to be escaped to render,
        so we do that instead of just the `#` -
        https://developer.microsoft.com/en-us/microsoft-
        edge/platform/issues/7157459/
*/
.todo-list li .toggle + label {
    background-image:
        url('data:image/svg+xml;utf8,%3Csvg%20xmlns%3D%22http%3A//www.w3.org/2000/svg%22%20width%3D%2
        22-10%20-
        18%20100%20135%22%3E%3Ccircle%20cx%3D%2250%22%20cy%3D%2250%22%20r%3D%2250%22%20fill%3D%22none
        width%3D%223%22/%3E%3C/svg%3E');


    background-repeat: no-repeat;
    background-position: center left;
}

/* circle checked - no checkmark when we click it! */
.todo-list li .toggle:checked + label {
    background-image:
        url('data:image/svg+xml;utf8,%3Csvg%20xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%22%20wi
        %22-10%20-
        18%20100%20135%22%3E%3Ccircle%20cx%3D%2250%22%20cy%3D%2250%22%20r%3D%2250%22%20fill%3D%22none
        width%3D%223%22%2F%3E%3Cpath%20fill%3D%22%233EA390%22%20d%3D%22M72%2025L42%2071%2027%20561-4%


}

/* NOTE: next is footer and its contents */

/* footer */
.footer {
    padding: 10px 15px;
    height: 20px;
    text-align: center;
    font-size: 15px;
    border-top: 1px solid #e6e6e6;
}

/* note the float here */
/* split into count, filters, and clear completed */
/* left, center, right */

/* left */
.todo-count {
    float: left;
}

/* center */
.filters {
    margin: 0;
    padding: 0;
    position: absolute;
    right: 0;
    left: 0;
}

.filters li {
    display: inline;
}

.filters li a {
    color: inherit;
    margin: 3px;
    padding: 3px 7px;
    text-decoration: none;
    border: 1px solid transparent;
    border-radius: 3px;
}

.filters li a:hover {
    border-color: #DB7676;
}

.filters li a.selected {
    border-color: #CE4646;
}

/* clear completed */
.clear-completed,
html .clear-completed:active {
```

```
    float: right;
    position: relative;
    cursor: pointer;
}

.clear-completed:hover {
    text-decoration: underline;
}
</style>
```

In the next section, we're going to learn about VueJS **Single File Components** (SFCs) and get started building our todo app.