Welcome!

Presentation Slides:
https://goo.gl/5oEeWd

Web Development Workshop: Introduction

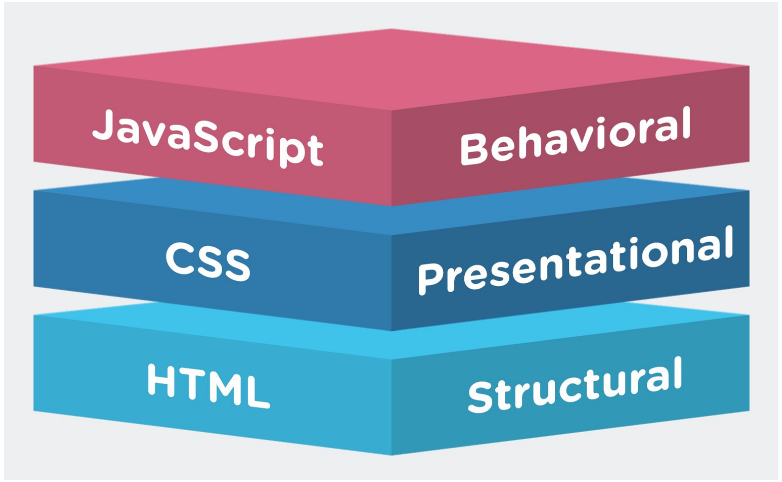
Ari Koumis
Hosted by Software and Computer Engineering Society (SCE)
sce.engr.sjsu.edu

Introduction:

- Workshop Overview
  - How web development works
  - Overview of text editors
    - Today we will use Sublime.
  - Basic introduction to HTML, JS, and CSS
  - End goal: Simple to-do list.

3 Core Languages in Web Development

- HTML
  - Creates of all the elements on the page
- JS
  - Provides the page with logic
- CSS
  - Makes the page look pretty



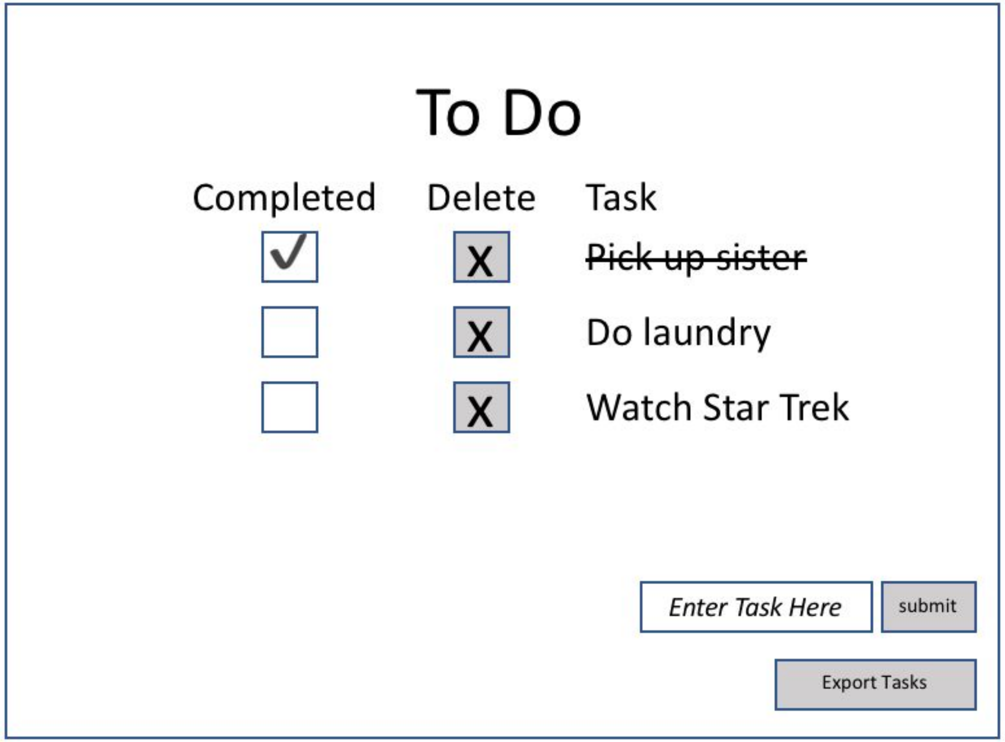
Simple Web Development

- 1000’s of Approaches to Web Development
  - Don’t confuse yourself - keep it simple!
- Today: Attempt “Vanilla” Web Development
  - Minimal use of external libraries
  - When using other libraries, always question: “Do I <u>need</u> this library?”

Project - To Do List

- Functional Requirements
  - Website will be static
    - View HTML file by opening it in browser.
  - Ability to add tasks
  - Ability to remove tasks
  - Check off tasks
- Non-functional requirements
  - Must look pretty

Project Mockup



If you get lost in the workshop:

- Project steps are on github:
    - https://github.com/ariskoumis/sce-webdev-workshop
- Bottom of slides will have the project at each step
    - Can either use git to clone the repo, or copy-paste the files into your folder.
    - Look for this logo:

Let’s Talk: Text Editors

- “Javascript is interpreted at runtime by the client’s browser”
  - You don’t compile anything yourself!
- What does this mean?
  - You don’t need an environment to compile client-side JS.
    - No equivalent to Eclipse needed
  - Can choose almost any text editor.
- My recommendations?
  - Sublime
    - Very straightforward, lightweight
    - **What we’ll use today**
  - Visual Studio Code
    - Built-in terminal
    - Built-in version control (e.g. git)
    - Syntax Highlighting

Let’s start! - HTML

1. Download Sublime
2. Download Google Chrome
3. Create a folder
4. Open that folder in Sublime
5. Use Sublime to create “index.html” in that folder

Double click index.html to view the contents in your browser!

HTML Overview

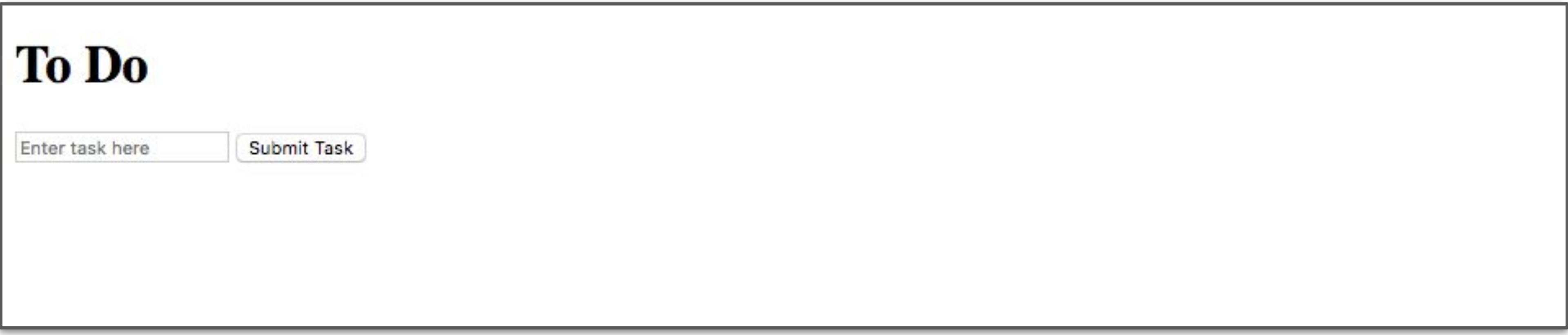
- Important Terminology
  - Elements - Contents of Webpage
  - Properties - Characteristics of Elements
- Example: Username Input Field

```
<input type="text" id="task-input" placeholder="Enter task here">
```

  - Element: Input
  - Properties
    - Type: Text Input
      - Specify what type of input
    - ID of Element: Task
      - Used to identify element in JS code.
    - Placeholder Text: Enter Task Here

Adding Elements

- Give HTML Page a Title
- Add a header, a text input box, and a button.



https://github.com/ariskoumis/sce-webdev-workshop/tree/4197eebb9094c63a00b403958b3c74459176331c

HTML Element - Div

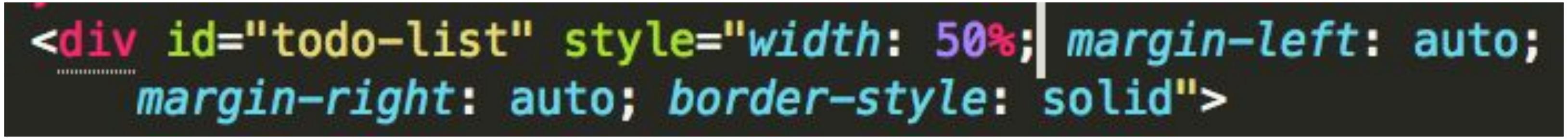
- div: A container for HTML elements
    - Useful for grouping things in a concise manner

- Let’s group our entire list into one container.

Styling the Div

- We want to center the list on our page and enclose it with a border
- Add the “style” property to our div element
    - Set width to 50%
    - Set margin-left and margin-right to auto
        - Will balance the div directly in the middle
    - Set border
        - Border-style (required)
            - Solid
        - Border-width (optional)
        - Border-color (optional)

Gross!



```
<div id="todo-list" style="width: 50%; margin-left: auto;
     margin-right: auto; border-style: solid">
```

- Hard to read, gets worse with more styling
- Style element is available, but try to avoid it.
    - Want to separate HTML elements from their styling
- Can give HTML elements from CSS (Cascading Style Sheets) files
    - CSS = Language that styles HTML elements

Creating CSS file

1. Using sublime, create file “style.css”
2. Copy your styling from the div, and paste it in style.css
    a. Remove “style” property from div element
3. Add identifiers
    a. #: identifies by ID
    b. .: identifies by Class

```
#example-div {
        width:100%;
        text-color: red;
}
```

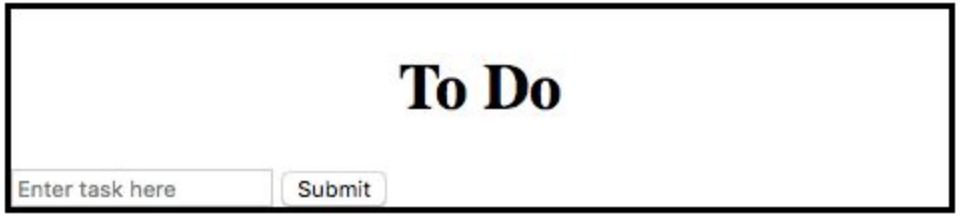
Link HTML to CSS file.

- HTML Element - <link/>
  - Place this element in the Footer of the HTML File
    - Why? - Links should run after HTML elements are present on page.

https://github.com/ariskoumis/sce-webdev-workshop/tree/c6c47cf1a21f8f0f1dc0a07e52fe4e27ffd76689

More CSS

- Center the header using *text-align*



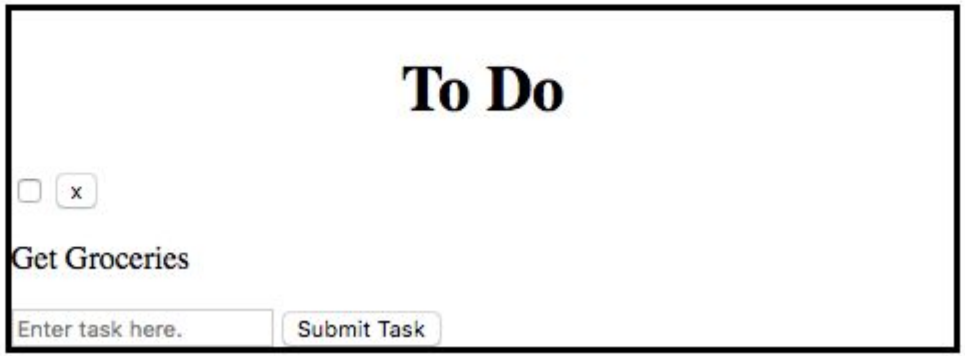
Add one task to HTML

- Will add one task to HTML file for reference
- End Result: Add HTML Dynamically

What does a task have?

- Status (Complete or Incomplete)
- Delete Button
- Task Name

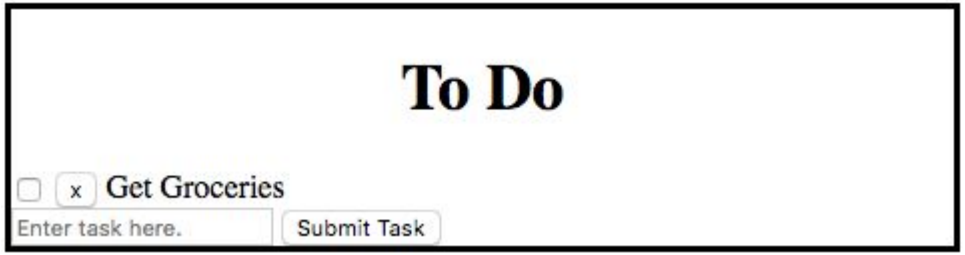
Bad Formatting!


To Do

Get Groceries

- All 3 elements should be in one line!
- Wrap all 3 in one div, and use the “span” element for the text

Much better.



To Do

x Get Groceries

https://github.com/ariskoumis/sce-webdev-workshop/tree/5d1ecc95fb03f9e8a83f05e551d85ab1f7aa049a

Adding Javascript

- Let’s add logic!
    - When we click the button, print the value inside of the task input box.

1. Using Sublime, create file “index.js”

Good Javascript Practice:

```
function main() {

        // do some stuff...

}



document.addEventListener(“DOMContentLoaded”, main);
```

- What does this do?
    - Only run JS code once DOM elements have all loaded
    - Prevents JS code from referencing elements that don’t exist yet.

Console.log()

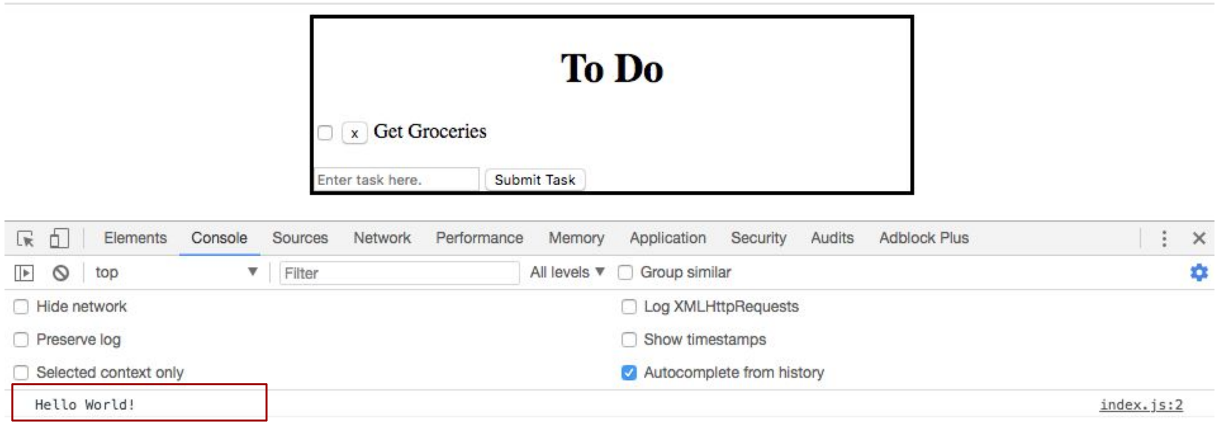
- Prints whatever you pass as a parameter to the console.
- Console is located in browser
    - In Chrome, right click and click “Inspect”

- Inside our main function, let’s print “Hello World!” to our console.

Link HTML to JS File

- Recall: For linking CSS we used a <link> element
- For JS: Use <script> element
- Also place it in the footer.

JS file is linked!



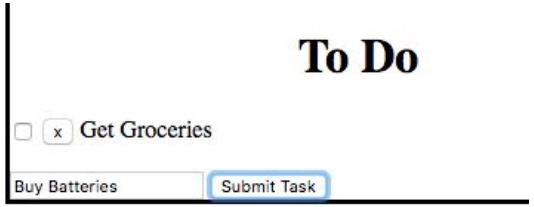
```
Hello World!                                                        index.js:2
```

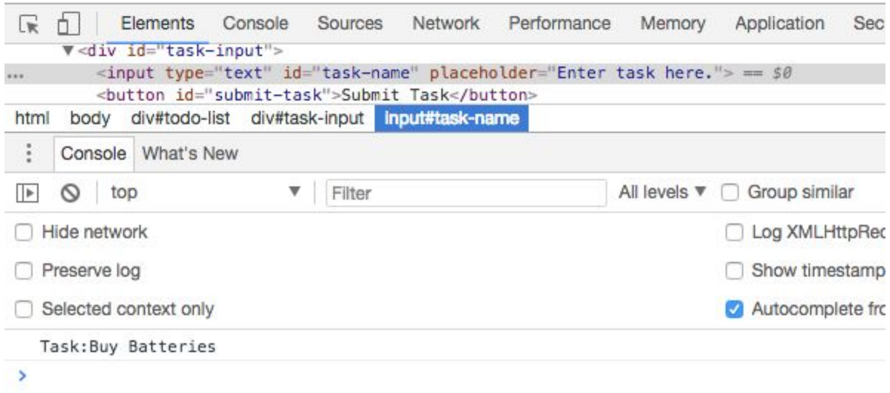
https://github.com/ariskoumis/sce-webdev-workshop/tree/d272f7ecf28c11c60f6e64ca9f15f61e853c00aa

Print: New Task’s Name when “Submit” pressed

- Clicking a button is an “event”
- element.addEventListener(“event-type”, callbackFunction)
  - Allows us to trigger functions when events occur
- Steps
  - Find “Submit Task” button
    - document.getElementById(“element-id”)
  - Attach event listener to button
  - Print text input’s value.

Wow!!!!





```
▼<div id="task-input">
    <input type="text" id="task-name" placeholder="Enter task here."> == $0
    <button id="submit-task">Submit Task</button>
```

```
Task:Buy Batteries
```

https://github.com/ariskoumis/sce-webdev-workshop/tree/c3a98e240bce87a353
1fb59cc3d295f71a3fda94

Feature: Add Task to List when Submit is clicked

- Really easy ways to do this with frameworks
  - React, MongoDB, Angular
- But we’re doing “vanilla” web development!!!
- General Approach
  - Maintain array of tasks
  - When submit button is pressed, render each task in a html element
  - Insert each html element into the “task-list” div.

“Injecting” HTML

- Can reassign an HTML element's content with:
    - element.innerHTML = “…”
    - “...” must be **HTML elements in a string**
- Use this with document.getElementById() to change any HTML elements
    - Let’s test this with simple injection.

- Every time a new task is submitted, inject the following

```
<div id="task-1">
		<input type="checkbox" id="task1-status"/>
		<button id="task1-delete">x</button>
		<span id="task1-name">Get Groceries</span>
</div>
```

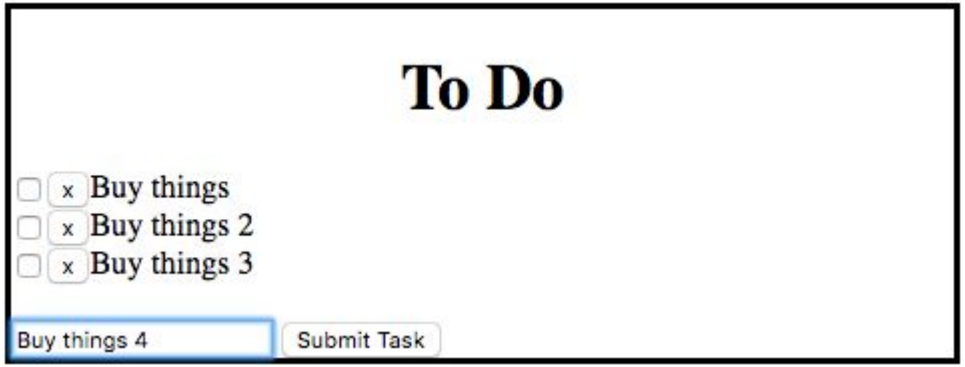
- Have to dynamically create this element in Javascript!

String Formation with Variables

- Two options
  - Classic Concatenation
    - var string = “value: “ + value;
  - Template Literals
    - var string = `value: ${value}`

- Template Literals **are the way to go!**

Working Task Addition


To Do

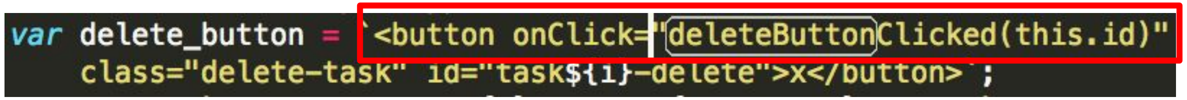
- x Buy things
- x Buy things 2
- x Buy things 3

Buy things 4

https://github.com/ariskoumis/sce-webdev-workshop/tree/6247b497f137bdb7194
3f5111bbf8040b7168752

Feature: Delete task from list

- General Approach
  - When a task's delete button is pressed, remove task from our array
    - Add "onClick" attribute to injected button
  - Regenerate div "task-list" by calling renderList();



```
var delete_button = `<button onClick="deleteButtonClicked(this.id)"
    class="delete-task" id="task${i}-delete">x</button>`;
```

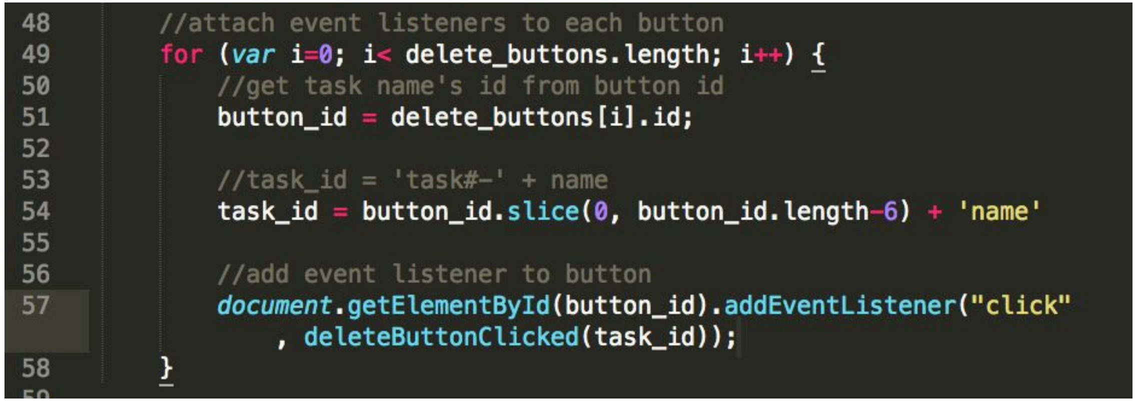
How do we get task’s name

- We have a rigid identification scheme - very useful
  - Checkbox: “task#-status”
  - Delete Button: “task#-delete”
  - Task Name: “task#-name”
- document.getElementsByClassName(“delete-task”) returns an array of all delete button elements
  - Each element in the array has “id” property
- To get task’s name
  - Get button’s ID
  - Swap“delete” from end of ID with “name”
    - Will use javascript .slice() function
  - document.getElementById(“task#-name”).innerHTML

Javascripts String.slice(beginning, end)

- String.slice(beginning, end) returns a substring
  - Ex:
    - string = “Hello World”
    - string.slice(0, 5) = “Hello”
    - string.slice(6,) = “World”
- We’ll use this to get the task’s name

Adding Event Listener to each button



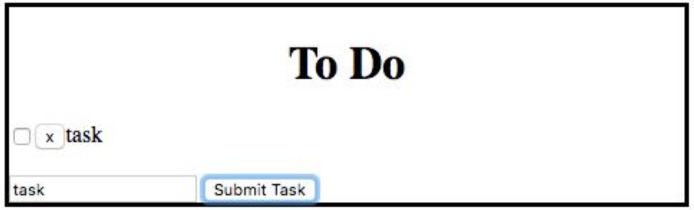
```
    //attach event listeners to each button
    for (var i=0; i< delete_buttons.length; i++) {
        //get task name's id from button id
        button_id = delete_buttons[i].id;

        //task_id = 'task#-' + name
        task_id = button_id.slice(0, button_id.length-6) + 'name'

        //add event listener to button
        document.getElementById(button_id).addEventListener("click"
            , deleteButtonClicked(task_id));
    }
```

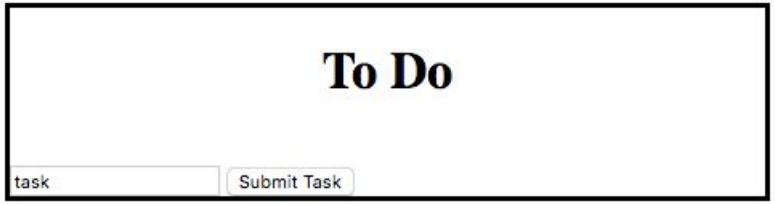
- Now we need to create deleteButtonClicked(task_name_id);

deleteButtonClicked(task_name_id)

- What it does:
    - Get task’s name
- Remove that task from our “tasks” array
    - *array.indexOf(string)* gives index of string in array.
    - array.splice(index, n) removes n indices from array, starting at the index.
- Regenerate tasks list
    - Can just call renderList() that we made earlier!

Deletion Works!

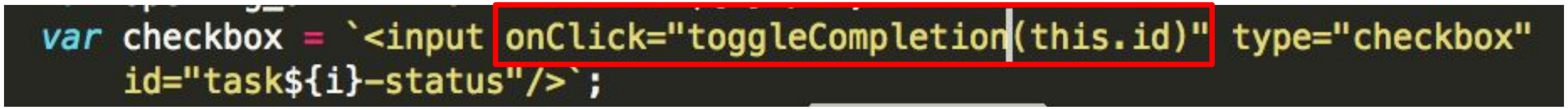




https://github.com/ariskoumis/sce-webdev-workshop/tree/4aee030cd3f09252905
a292a4cb60ff132e3edab

Final Feature: Strikethrough Completed Tasks

- General Approach
  - Add onclick function to injected checkbox
  - Function toggles “strikethrough” style on task name.

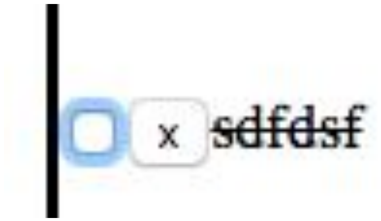


```
var checkbox = `<input onClick="toggleCompletion(this.id)" type="checkbox"
    id="task${i}-status"/>`;
```

How do we set styles of elements?

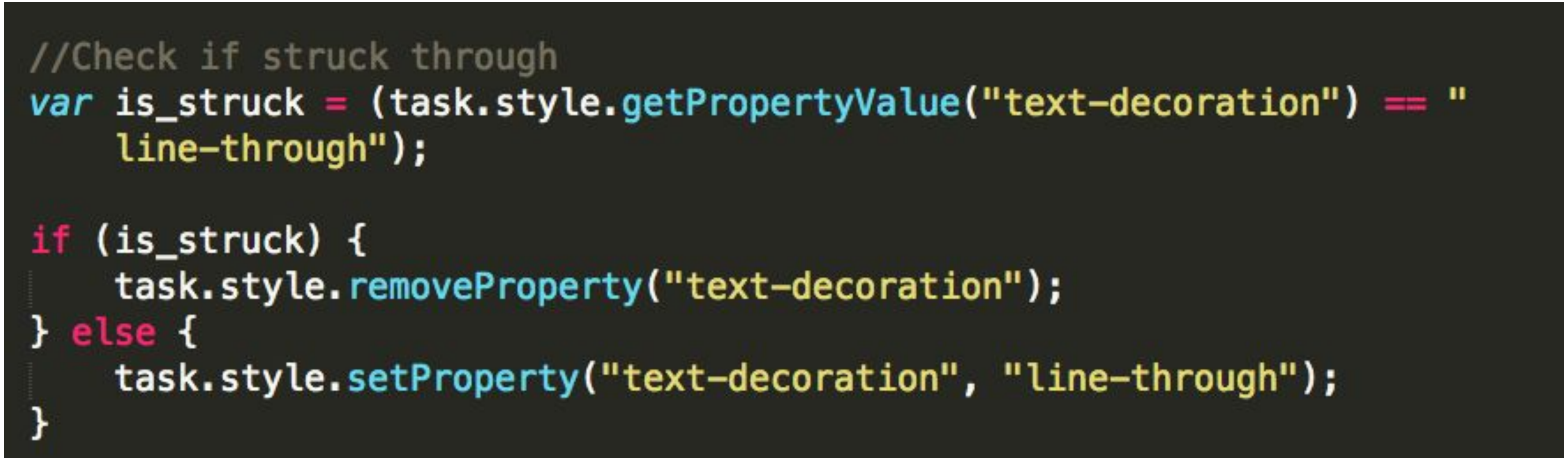
- element.setProperty(attribute, value)
  - We will use it like so:
  - element.setProperty(“text-decoration”,”line-through”);
    - Remember: have to get the Element first!
- To strikethrough, we set the following styling element:
  - Text-decoration: line-through

Checks, but doesn't uncheck


x ~~sdfdsf~~

- toggleCompletion() needs to check if it's struck through.
    - element.getProperty(property_name)

How are we handling strikethroughs?



```
//Check if struck through
var is_struck = (task.style.getPropertyValue("text-decoration") == "
    line-through");

if (is_struck) {
    task.style.removeProperty("text-decoration");
} else {
    task.style.setProperty("text-decoration", "line-through");
}
```

Done!

- Questions?

https://github.com/ariskoumis/sce-webdev-workshop/tree/d05d141e5a3daf5df46
6eb2be9c7d51e18cef427