ADVANCED JAVASCRIPT

CoderDojo

SETUP

Card 1 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

1 This set of Sushi Cards will take you through creating a to-do list web-app. If you'd like to see a finished version at any point, you can check mine out at dojo.soy/js-todo. This app can be used to track whatever you want: cool programming tricks you want to learn, places to go, songs to listen to (or learn to play!), or something as simple as things to pick up at the shops.

2 To get started, download the zip file at dojo.soy/a-js-files and extract the zip file into a folder on your computer. You're going to do all the work for this web-app there.

3 You're only going to need to worry about two of the files in the folder:

- **index.html** — The web page your JavaScript will be running on, which you'll want to keep open in a browser so you can see your changes as you work. Don't forget — you'll need to refresh after you've saved!
- **js/to-do.js** — The JavaScript file in which you'll be writing all your code. There's some **CSS** included too, to make the page look cool (and if you've done the **HTML/CSS** Sushi Cards, feel free to mess about with it), but you shouldn't **need** to change it or the version of **jQuery** that's included.

4 These are the **Advanced Sushi Cards**, so I'll sometimes tell you to do things without showing you how they're done. In those cases, they're almost always something that you can find in the **Beginner** or **Intermediate** JavaScript Sushi Cards.

CoderDojo

SETUP

Card 1 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

5 You're also coming to the point where you know enough JavaScript to learn by searching for answers online and looking at example code there. While I don't make you do that at any point in these cards, I do point out a few things it might be cool to look up. You should check them out!

6 Once you've finished with these Sushi Cards, you should work with your Dojo's mentors on coming up with new projects that interest you. If you don't know it yet, you could also look at the **HTML/CSS Sushi Cards** to learn two more languages that work really well with JavaScript.

7 Good luck! Have fun!

ADVANCED JAVASCRIPT

CoderDojo

PROJECT OUTLINE & WIRING BUTTONS

Card 2 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

1 Before getting into the details of how your application is going to work, you should connect all the buttons up to "dummy" functions, to make testing them easier.
From the **Intermediate JavaScript Sushi Cards**, you will hopefully remember that a JavaScript **function** definition looks like this:

```
function newToDoItem(itemText, completed) {
   console.log("New item created: "+itemText+"
Completed state: "+completed);
}
```

2 Almost all the **functions** you need are in the file already, though they don't do anything cool yet. They just send **console.log** messages telling you what **function** was called and what the values in the parameters were. Read through them to understand what's there.

3 There's one missing, that you need to add. Create a new function in **to-do.js** that:

- Is called **toggleToDoItemState**
- Accepts a parameter: **listItem**
- When called, logs the message **"Toggling state of item "+itemId**

CoderDojo

PROJECT OUTLINE & WIRING BUTTONS

Card 2 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

4 To connect things you're going to use **jQuery** (also from the Intermediate Cards!). Add the following code at the end of **to-do.js**:

```
$(document).ready(function(){
    alert("Ready to go!");
});
```

Now load **index.html** in your browser. You get an alert when the page loads. This is because when the **document** (index.html) was **ready** (loaded), the **function** above (which contained the **alert**) was run.

5 Now you need to tell JavaScript to **listen** for a **click** on the buttons, then do something. Each of the buttons on the page has a unique **ID**. You can use that **ID** to tell **JavaScript** what to listen to. Add the following code instead of the **alert** in the **document.ready function**:

```
$("#add-button").click(function(){
    addToDoItem();
});
```

Reload **index.html** and click the button that says **Add**. You should see a message in your **console**. Do the same for the other two buttons:

- Connect **#clear-button** to **clearCompletedToDoItems**
- Connect **#empty-button** to **emptyList**

ADVANCED JAVASCRIPT

CoderDojo

CREATING A LIST

Card 3 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

1 Right now, the to-do list on the page is just some HTML that I put in there so you'd have something to look at! Time to make it yours: Go into **index.html** and delete the four lines that start with **li** tags. If nothing's changed, they'll be lines 23–26. Then reload **index.html** in your browser. It looks a little empty, but you'll fix that soon!

2 Inside the **document.ready** function, call the **loadList** function. Reload **index.html** to check that it's being called. If you got the log in your **console** then you are all set!

3 Now you need to make the **loadList** function actually load a list! For now, it's going to be a sort of a "demo list". Later, I'll show you how to save a to-do list and load it when you re-open the page (though you will need to be online for that). So, change the contents of the **loadList function** to:

```
toDoItems = [
  {
    "text": "My",
    "completed": false
  },
  {
    "text": "to-do",
    "completed": true
  },
  {
    "text": "list",
    "completed": true
  }
];
```

CoderDojo

CREATING A LIST

Card 3 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

4 You can see that these three items each have their **text** and their **completed** status and that they're stored in an **array**. However, you still need to put them on the page. Inside the **getToDoItemHTML** function, add this bit of code:

```
var itemHTML = $("<li>"+toDoItem.text+"</li>");
if(toDoItem.completed === true){
 $(itemHTML).addClass("completed");
}
return itemHTML;
```

It uses an **if** statement, which makes a **jQuery** object of HTML and, **if** the **toDoItem** is completed, adds the "completed" **class** to it for styling.

5 Now you can create the HTML for a list item, you need to add it to the list. Add this to the **loadList function**, after the code from step 3:

```
toDoItems.forEach(function(toDoItem){
   var itemHTML = getToDoItemHTML(toDoItem);
   $(".todo-list").append(itemHTML);
});
```

This code uses **forEach**, which goes through every item in the **array** and **appends** it to the to-do list, which you selected by its **class** name.

ADVANCED JAVASCRIPT

CoderDojo

ADDING AN ITEM TO THE LIST

Card 4 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

1

So, your to-do list works now, but it's always the same. You can fix that! The "Add" button already calls a **function**, you just need to make that **function** do something. Go to your JavaScript file and change the code in **addToDoItem** to get the text from the text field (named **"new-todo"**) and update the log to check you're getting it:

```
var itemText = $("input[name=new-todo]").val();
console.log("Item added: "+itemText);
```

val( ) in this case is short for value. Type something into the text field and click the "Add" button. You should see what you typed in the **console**.

2

The next step is to get that text onto the to-do list. You'll need a **function** that gets the HTML for the new item and adds it to the end of the list. So, change the **newToDoItem** to have this code inside it:

```
 var newItem = {
                  "text": itemText,
                  "completed": completed
              }; // make parameters into a toDoItem
 toDoItems.push(newItem); //put newItem at the end
of toDoItems
 var itemHTML = getToDoItemHTML(newItem);
 $(".todo-list").append(itemHTML);
```

CoderDojo

ADDING AN ITEM TO THE LIST

Card 4 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

3 Now, go back to **addToDoItem** and change it again, removing the log (if you like) and calling **newToDoItem** like this:

```
newToDoItem(itemText,false);
```

Don't repeat yourself!

You can also use **newToDoItem** in **loadList** to reduce the amount of code you've got to manage. You can replace all the code in the function with this:

```
newToDoItem("My",false);
newToDoItem("to-do",true);
newToDoItem("list",false);
```

You've gone from 17 lines of code to 3, and that's the power of functions! Try to understand how this works and how it's doing the exact same job as the old version of **loadList** was.

ADVANCED JAVASCRIPT

CoderDojo

MARKING AN ITEM COMPLETE

Card 5 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

1

Now you can load your to-do list and add items to it, but what's the point if you can't check them off when you've done them? Next up, you're going to do just that, and it'll be pretty easy. First, you need to connect double-clicking on a list item (**li** tag) to your **toggleToDoItemState** function. You can do this by having **jQuery** listen for double-clicking on **li** tags and running a function that calls your **toggleToDoItemState** function. It looks like this:

```
$(document).on("dblclick","li", function(){
   toggleToDoItemState(this);
});
```

What's happening here is:

- jQuery is listening throughout the whole page for **dblclick** (double-clicking) on **li** (list item) tags.
- When jQuery hears a double-click on a list item, it runs **toggleToDoItemState** and passes it **this**, which is a special **keyword**

So what's this all about?

In JavaScript **this** is a **keyword**, a word with a special meaning. Exactly how it works varies depending on where it's used (you can learn more online — dojo.soy/a-js-this but in this case it means **this** list item, the one that was double-clicked.

CoderDojo

MARKING AN ITEM COMPLETE

Card 5 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

2 Next, you need to update your **toggleToDoItemState** function so it does two things:

- Updates the array of to-do items
- Adds the **completed** class to the item that was clicked First, update the array by replacing the code in **toggleToDoItemState** with this:

```
var itemId = $(listItem).index();
toDoItems[itemId].completed =
!toDoItems[itemId].completed;
```

The first line uses jQuery to get the **index** of the **li** inside the ordered list (**ol**) tag that it is inside. The **index** is the number that indicates its position and, since computers count from 0, it will be one lower than the number shown on the web page. The second line looks up that item in the array (the **index** is the same as on the list, because we don't move the items around) and sets its **completed property** to the opposite of the current value. Putting **!** in front of a **true**/**false** value turns it into its opposite.

3 To change the class and give you a crossed-out item on the list, all you need to add is one more line at the end of **toggleToDoItemState**:

```
$(listItem).toggleClass("completed");
```

CoderDojo

CLEANING THE LIST

Card 6 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

1 If your user wants to empty their to-do list (say they've been playing with it and added a lot of stuff that's not real), you can let them do that. It's pretty easy actually! All you need to do is empty the to-do array in the JavaScript and the ordered list in the HTML. You can empty the array by updating the **emptyList** function to be the following:

```
toDoItems = [];
```

Thats it! **toDoItems** is now an empty array!

2 Now, to empty the **ol** in the HTML isn't much trickier. Just add the following code into the end of your **emptyList** function:

```
$(".todo-list").children().remove();
```

This tells jQuery to:

- Select all tags with the class **todo-list**. In your case there's only one, the **ol**. Be careful using this if you've used the class in several places.
- Then select all the **children** of that tag. That means every tag inside it, in this case all your **li** tags.
- Then **remove** all the tags it's selected from the page.

CoderDojo

CLEANING THE LIST

Card 6 of 6

I'm Learning: JavaScript

Produced by CoderDojo Foundation // @CoderDojo

3 Now, what about those crossed-out items? It's going to get very messy if you don't let users clean them up too. Start with removing the completed items from the array. Update **cleanupCompletedToDoItems** like this:

```
toDoItems = toDoItems.filter(function(toDoItem){
return !toDoItem.completed;
});
```

This code takes the **toDoItems** and runs a **filter** on them. Only **toDoItems** that return a **true** value from the filter function will pass and be in the new **array** that's created (which you just assign back into **toDoItems**). By taking the **completed** value of each **toDoItem** and making **true** turn into **false** and **false** turn into **true** with **!**, you get only those items that are not completed out.

4 Finally, remove the completed items from the list on the HTML page by adding one more line to your **cleanupCompletedToDoItems** function, like this:

```
$(".todo-list").find(".completed").remove();
```

This does something very similar to the code in 2 above, except you don't select all the children of the **.todo-list**, you just **find** the ones with the **completed** class and **remove** them.