Table of Contents

1 Getting Setup 1.1

2 Wiring up buttons 1.2

3 Creating a list 1.3

4 Adding to-dos 1.4

5 Completing items 1.5

6 Cleaning the list 1.6

1 In these cards you're going to make the to-do app you can see 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 These are the Intermediate 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 see in the Beginner JavaScript Sushi Cards.

3 Go to dojo.soy/trinket and click "Sign Up For Your Free Account" if you do not already have an account. You will need an email address to sign up.

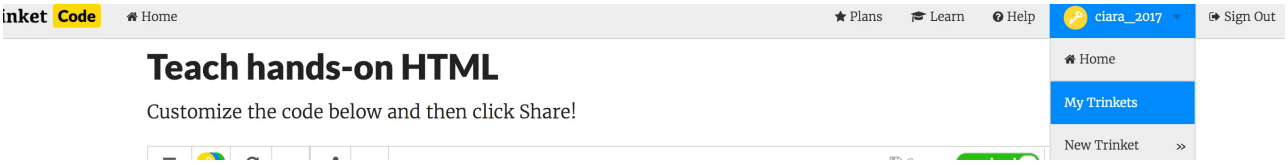
4 Enter your email address and choose a password, or ask somebody to do this for you.

5 Creating an account allows you to save your work and access it from any computer. It also allows you to make a copy of a project somebody else has shared with you so you can make your own changes to it!

6 Go to dojo.soy/js-i-start. You will see a box containing an example JavaScript website project. On the right hand side is the website, and on the left hand side is the code that makes the website. If you are not signed in, you will need to enter your email address and password to be able to **Remix** the project.

7 Click the "Remix" button at the top right of the project (if it is not green, you have to sign in and then click it again). This creates a copy of the project for you to work with. It should say "remixed" after you click it

8 Next to the "Sign Out" button at the very top right corner of the page you should see your username and a drop-down menu (the tiny triangle tells you there is a drop-down). Click on it to show the menu and then select "My Trinkets".



In Trinket (this website), projects are called "Trinkets"

9 The project you just remixed will be shown together with some example projects for other programing languages. It will be called "Intermediate JavaScript Remix". Click on it to begin editing!

10 You should also click on the drop-down arrow beside the “Autorun” button and select “Click To Run” instead. This makes sure that the code will only run when you click this button, not when you're halfway through typing it!

11 You're only going to need to worry about two of the files in the Trinket:

- `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!
- `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.

12 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!

13 Good luck! Have fun!

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 *Beginner JavaScript Sushi Cards*, you will hopefully remember that a JavaScript **function** definition looks like this:

```
function newToDoItem(itemText, completed) {
    alert("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 **alert messages** (pop-up boxes) 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, alerts the message `"Toggling state of item "+itemId`

4 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` into the `runWhenPageLoads` function:

```
document.getElementById("add-button").addEventListener("click", addToDoItem)
```

Because `runWhenPageLoads` is called at the end of the file, and the file is loaded at the end of the HTML page, `runWhenPageLoads` will, as you might guess, run once the page loads! This bit of code chains together a lot of functions with dots (`.`), where each function's is acting on the thing fetched by the function before it:

- Look in the `document` —the whole page
- Find the **element** (HTML tag) on that page that has “add-button” as its `id`
- Tell JavaScript to **listen** for the **click** action on the button and then run the `addToDoItem` function whenever it “hears” the action

Reload `index.html` and click the button that says “Add”. You should see an **alert** message pop up! Do the same for the other two buttons:

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

Don't forget to test them!

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 `runWhenPageLoads` function, call the `runWhenPageLoads` function. Reload `index.html` to check that it's being called. If you got the **alert** then you're 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. First, let's talk about lists in JavaScript. Lists, or **arrays** as they're called in JavaScript, are groups of the same kind of **variable** (number, text, etc.) in order. They don't need individual names, since you can look them up from the **array**. You use square brackets (`[]`) to create an **arry**. They're also a kind of **variable**, so you can give them names like any other **variable** (and yes, you can have **arrays** of **arrays**, but we're not doing that just yet!). Change the contents of the `loadList` to create an empty **array** called `toDoItems` like this:

```
toDoItems = [ ]
```

4 You now need to add some things to that **array**. There's a problem, though: a to-do has two parts! You need to know what it is (a **string** of text) *and* whether or not it has been completed yet (a true/false value). Because the need to know if things are true or false is so common (is the user signed in? does the player have more than 10 points? etc.) JavaScript has a special kind of variable for true/false values. It's called a **boolean** and it can only be either `true` or `false`. Now you know what you need, make up three to-do items for your list and decide which one you'll mark done (to show the user they can do that).

5 So, with three to-dos to add to your **array**, how do you do that? You can create an **object variable** in JavaScript that contains related **variables**. In the case of your to-dos that will be their `text` and their `completed` state (either `true` or `false`). Here's an example object:

```
exampleToDo = {
              text: "Buy milk",
              completed: false
              }
```

The **object** is wrapped in curly braces (`{ }`) and stores pairs of **keys** (`text`) and **values** (`Buy milk`). Now it's time to put your **objects** into your **array** to make a list you can load when the page does!

6 So, change the contents of the `loadList` function like this (but use the to-dos you made up!):

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

7 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 listItem = document.createElement("li")
var itemText = document.createTextNode(toDoItem.text)
listItem.appendChild(itemText)

if(toDoItem.completed === true){
 listItem.classList.add("completed")
}

return listItem
```

This code:

- Creates a piece of HTML (a `li` or list item element) and stores it in the `listItem` variable.
- Then an `itemText` variable is created and a `TextNode` (the bit of text that'll sit inside the HTML tag) is stored in it with the `text` of the `toDoItem` that was passed into the function.
- Then it adds ("appends") the `itemText` inside the `listItem` element.
- If the `toDoItem` 's `completed` property is `true` , then it adds a `class` to the item. This tells the HTML page to display the completed task differently (based on some code I wrote for you). If you've done the *HTML/CSS Sushi Cards* then you can take a look at the CSS files and edit what they look like if you want!

8 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 **array** of items.

```
toDoList = document.getElementById("todo")


toDoItems.forEach(function(toDoItem){
   itemHTML = getToDoItemHTML(toDoItem)
   toDoList.append(itemHTML)
})
```

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

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 `to-do.js` file and change the code in `addToDoItem` to get the text from the text field (named “new-todo”) and alert it to check you're getting it:

```
itemText = document.getElementById("new-todo").value
alert("Item added: "+itemText)
```

Type something into the text field and click the "Add" button. You should see what you typed pop up in an **alert**.

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); //use the push function: put newItem at the end of toDoItems
var toDoList = document.getElementById("todo")
var itemHTML = getToDoItemHTML(newItem)
toDoList.append(itemHTML)
```

3 Now, go back to `addToDoItem` and change it again, removing the **alert** 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.

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 JavaScript **listen** for double-clicking on `li` tags and running a function that calls your `toggleToDoItemState` function. It looks like this:

```
document.getElementById("todo").addEventListener("dblclick",function(event){
 if(event.target && event.target.matches("li")){
   toggleToDoItemState(event.target)
 }
})
```

What's happening here is:

- JavaScript is **listening** to all double-clicks that happen on the `todo` list
- If it “hears” one, it runs a **function**, into which it passes the details of the click `event`
- If the `event` had a `target` (a thing that was clicked on) and that `target` matches `"li"` , which is the HTML tag for a **list item** then:
  - Run `toggleToDoItemState` and pass it the `event.target` information

2 Next, you need to update your `toggleToDoItemState` function so it adds the `completed` **class** (a HTML property—a **tag**, like `<li>` can have many **classes**) to the item that was double-clicked, which will put a line through it using some of my CSS code. Thankfully, in modern JavaScript, that's pretty simple! Just update this to `toggleToDoItemState` :

```
function toggleToDoItemState(target){
 target.classList.toggle("completed")
}
```

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 ordered list in the HTML. To empty the *ol* in the HTML isn't too tricky. Just add the following code into the end of your *emptyList* function:

```
document.getElementById("todo").innerHTML = ""
```

This tells JavaScript to get the to-do list and remove all the HTML inside it. That removes all the `<li>` tags, which are the to-do items!

2 Now, what about those crossed-out items? It's going to get very messy if you don't let users clean them up too. They all have the same **class**— `completed` —so you can use that to pick out the items to remove by selecting based on **class** instead of **id**. Change `clearCompletedToDoItems` like this:

```
function clearCompletedToDoItems() {
 var items = document.getElementsByClassName("completed")
 for(var i=0; i<items.length; i++){
   items[i].remove()
 }
}
```

This code is slightly tricky, but it's getting an a list of all the HTML **elements** with the `completed` **class** and then using a `for` loop