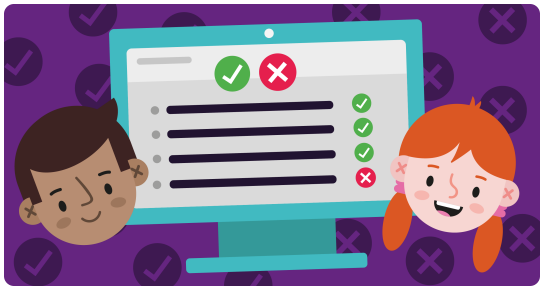
To-do list

Code a to-do list web app to store important notes



Step 1 Introduction

With these cards you're going to make a to-do list web app. You'll be able to use this app to track whatever you want: cool programming tricks you want to learn, places to go, songs to listen to (or learn to play!), or just something as simple as things to pick up at the shops.

What you will make

This is an example of the app you'll be making:

You can add and remove to-do items from the list, and also save them so they reload for you later.

What you will learn

- Using an online editor to create and modify JavaScript
  - Creating your own JavaScript functions
  - Listening for different user actions on a web page
  - Using functions together to write code more professionally
  - Saving user information between visits to your web page

What you will need

Hardware

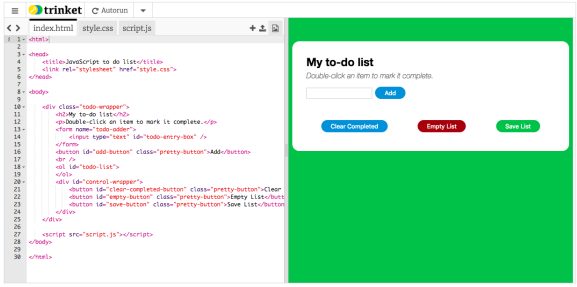
- A computer capable of accessing trinket.io (https://trinket.io)

Software

This project can be completed in a web browser using trinket.io (https://trinket.io).

Step 2 Get set up

- Go to the starter trinket (http://dojo.soy/js-i-template). You will see a box containing an example website project. On the right-hand side is the website, and on the left-hand side is the code that makes the website.



Let's start coding!

Step 3 Add action to your buttons

The code you've got includes three files, which you can see as tabs in the trinket window:

- `index.html` — a HTML file that tells the page what should be on it
- `style.css` — a CSS file that tells the page what it should look like: where things should be, what size and colour they should be, etc.
- `script.js` — a JavaScript file that tells the page what to do; you'll be doing most of your coding in this file

If you look at the page, you'll see it has four buttons:

- An Add button for adding new to-do items
- A Clear Completed button for clearing items that you've marked as finished
- An Empty List button for completely emptying the to-do list
- A Save List button for saving what's on the list

Of course, since you haven't written any code yet, right now none of them do anything!

Since you want to make the page do something, you need to click on the tab for the `script.js` file and add some code in there. These instructions will show you how to set up the Add button, and then you can set up the others by yourself.

JavaScript needs to be told which parts of the HTML page are important, and which interactions of a user with these parts it should react to. In this case, you want to tell it about the Add button, and tell it to react when the user clicks this button.

Getting the button

Start by making a variable for the button and telling JavaScript to get the element from the HTML document that has the Id `add-button`.

```
var addButton = document.getElementById("add-button");
```

An Id is a unique label for a part of a web page, and when we created the starter page, we gave a label to each of the buttons. You can see them if you look at `index.html`.

Listening for the click

Now connect your button to a event listener, so JavaScript will 'listen' for a particular kind of event and then run a function when it 'hears' it. In this case, the event is a click. Do this with the `addEventListener` function, like this:

```
addButton.addEventListener("click", addToDoItem);
```

This listener will wait for a click on `addButton`, and when it 'hears' the click, it will react by running the `addToDoItem` function. Of course, it won't work just yet, since you haven't written an `addToDoItem` function yet!

Creating the function

Later in the project you'll be writing code for your functions so that they add to-do items, clear the list, save it, etc. But for now, you just want to check that you've connected your event listeners properly.

Create your `addToDoItem` function so that it will pop up an alert message telling the user which button they've clicked.

```
function addToDoItem() {
  alert("Add button clicked!");
}
```

Now click the button and check if it works!

Write code for the other buttons

Now connect the other three buttons so clicking them sends an alert:

Connect the Clear Completed button — which has the Id `clear-completed-button` — to an alerting function called `clearCompletedToDoItems`.

Connect the Empty List button — which has the Id `empty-button` — to an alerting function called `emptyList`.

Connect the Empty List button — which has the Id `save-button` — to an alerting function called `saveList`.

The code below the `addButton` section is what you need to add:

```
var addButton = document.getElementById("add-button");
addButton.addEventListener("click", addToDoItem);
function addToDoItem() {
    alert("Add button clicked!");
}

var clearButton = document.getElementById("clear-completed-button");
clearButton.addEventListener("click", clearCompletedToDoItems);
function clearCompletedToDoItems() {
    alert("Clear button clicked!");
}


var emptyButton = document.getElementById("empty-button");
emptyButton.addEventListener("click", emptyList);
function emptyList() {
    alert("Empty button clicked!");
}


var saveButton = document.getElementById("save-button");
saveButton.addEventListener("click", saveList);
function saveList() {
    alert("Save button clicked!");
}
```

Step 4 Add to-do items

Time to get the first of those buttons working properly! This step will show you how to make it add a to-do item to the list.

HTML lists

You're going to use a tiny bit of HTML in this step. The list is an ordered list — that means it's numbered. The HTML tag for an ordered list is `<ol>`, and each individual list item needs an `<li>` tag.

Adding list items

The page came with the `<ol>` ordered list, so you just need to write some JavaScript to add `<li>` tags for each new to-do item. The user should be able to enter text in the box on the page, and then click the Add button to see it appear on the list as a numbered item.

First, just like you did with the buttons, create variables to select the text box and the list. They already have the Ids `todo-entry-box` and `todo-list`.

```
var toDoEntryBox = document.getElementById("todo-entry-box");
var toDoList = document.getElementById("todo-list");
```

Now you can easily access the box and the list from inside your program.

Create a function called `newToDoItem` to add an item to the list. This function will need to know two things:

- What is the text of the item?
- Should the item be marked as completed?

Of course, no new to-do item would ever be complete, but you're planning ahead here: you'll be able to use the same function again when you're loading a saved list that has some completed items on it!

```
function newToDoItem(itemText, completed) {
    var toDoItem = document.createElement("li");
    var toDoText = document.createTextNode(itemText);
    toDoItem.appendChild(toDoText);

    if (completed) {
        toDoItem.classList.add("completed");
    }

    toDoList.appendChild(toDoItem);
    toDoItem.addEventListener("dblclick", toggleToDoItemState);
}
```

Now, connect to the function to the Add button: just change your `addToDoItem` function to get the text from the box and pass it to the `newToDoItem` function you've just created.


```
function addToDoItem() {
    var itemText = toDoEntryBox.value;
    newToDoItem(itemText, false);
}
```

Since a new to-do item is never complete, you can always pass `false` to the `completed` parameter of the `newToDoItem` function.

Now try adding a to-do to the list!

Step 5 Completing items

There's not much point to a to-do list if you can't mark items as done! Time to add that functionality.

You've already set up the listener for a double-click on a to-do item. All you need to do now is write a function that will toggle the item between complete and not complete when that double-click happens.

Remember that you're using the `complete` class to mark items as complete. Not having that class means they're not complete. So all your function needs to do is add or remove the class from the item's class list, either adding it if it's not on the list yet, or removing it if it is.

The this keyword

The trick is knowing on which item to toggle the class. To identify the item that was clicked, you'll need to use a new JavaScript keyword: `this`.

How exactly the `this` keyword works is a bit complicated, but all you need to know here is that, when it's used with a function called by an event listener, it means 'the element the listener was bound to'. So you can use `this` to identify the specific `<li>` item that was clicked!

Add the `toggleToDoItemState` function to your script like so:


```
function toggleToDoItemState() {
    if (this.classList.contains("completed")) {
        this.classList.remove("completed");
    } else {
        this.classList.add("completed");
    }
}
```

Step 6 Remove items

Once you've marked items as complete, you'll want a way to remove all those completed items. Also, if you come back to your list after a long time, or if you just want to work on something totally new, you might want to clear out everything on it. To do this, you just need to update two functions you've already connected to buttons: `clearCompletedToDoItems` and `emptyList`.

Clearing completed items

Just like you can select all the elements in an HTML document, you can select the elements inside any other element. Elements inside another element are called the children of that element. Likewise, just like you can select elements by Id, you can select them by class too.

In order to clear completed items, update the `clearCompletedToDoItems` function with code to select the children of `toDoList` (the items inside it) that have the `completed` class. Then loop over the selected items to remove them one by one.


```
function clearCompletedToDoItems() {
    var completedItems = toDoList.getElementsByClassName("completed");

    while (completedItems.length > 0) {
        completedItems.item(0).remove();
    }
}
```

You can see that the code always removes the item at list position `0`, the first item on the list. You need to use `0` to do this, because JavaScript starts counting at `0` and not `1`! You remove this item so that every time the loop runs, it removes the first item, so the list gets shorter and shorter. In this way, no matter how many completed items are on the list, the loop will eventually remove them all.

Clearing everything

To clear everything off the list, do the same thing as above, but select all the children of `toDoList`.


```
function emptyList() {
    var toDoItems = toDoList.children;
    while (toDoItems.length > 0) {
        toDoItems.item(0).remove();
    }
}
```

Step 7 Save the list

To make your to-do list even more useful, you can save it to the `local storage` on the user's computer. Then, as long as they open it in the same browser the next time, it will remember their to-do list!

There are two parts to this: saving the list and, if it's there, loading it again when the page is reloaded.

This gets a bit tricky: local storage can't store HTML, so you need to take the HTML code and turn it into pure JavaScript. To do this, you'll need an `array`.

Arrays

An array is a special kind of variable that's a list of variables. You can create one with square brackets `[]`, and add items to it with the `push` method. You can remind yourself what a specific array item is using `alert` and the item's position in the array. Remember that JavaScript starts counting at `0`!

```
var myArray = [];
myArray.push("something to store");
myArray.push("something else to store");
alert(myArray[0]);
//This will alert "something to store"
```

Next, you need to loop over the `toDoList` list and add each item to the array. Remember that you need to store not just the task, but also whether or not it's completed. The best way to do this is using JavaScript objects.

JavaScript objects

An object is set of properties and values. You create one like this:

```
var toDoInfo = {
    "task": "Thing I need to do",
    "completed": false
};
```

Once you've converted all the to-do items into objects, you just need to save them to local storage. Local storage can only store strings, but luckily JavaScript turns arrays into strings for you if you use the `stringify` function!

Time to try it all out!

Putting it all together

Update the `saveList` function to:

- Make an array
- Use a `for` loop to put every item in `toDoList` into the array as an object
- `stringify` the array and store it in local storage with the key `toDos`

```
function saveList() {
    var toDos = [];

    for (var i = 0; i < toDoList.children.length; i++) {
        var toDo = toDoList.children.item(i);

        var toDoInfo = {
            "task": toDo.innerText,
            "completed": toDo.classList.contains("completed")
        };

        toDos.push(toDoInfo);

    }

    localStorage.setItem("toDos", JSON.stringify(toDos));
}
```

Step 8 Load the saved list

To load the list, you need to reverse everything you did to save it. But first, you need to check if there's anything to load. You do this by checking if the key you used to store the list doesn't have a `null` value. 'Null' is just another word for 'empty', or 'nothing'.

Create a `loadList` function and have it:

- Check if the `toDos` key exists in local storage
- If it does, load it into a variable as an array
- Loop over the array, and use `newToDoItem` to create new to-do items for everything in it

```
function loadList() {
    if (localStorage.getItem("toDos") != null) {
        var toDos = JSON.parse(localStorage.getItem("toDos"));

        for (var i = 0; i < toDos.length; i++) {
            var toDo = toDos[i];
            newToDoItem(toDo.task, toDo.completed);
        }
    }
}
```

Call the `loadList` function after you've created it.


```
loadList();
```

Challenge: example to-do items

See if you can make the `loadList` function create some example to-do items if there aren't any saved.

Step 9 Challenge: automatic saving

Change the code of your app so that, instead of the user having to click the Save button, their changes to the list are automatically saved.

You won't need to create any new functions to do this, but you will need to change several you already have!
