Introduction

JavaScript course

By

Fahad Alshehri

what we will learn

- comments
- The Three ways to declare a variable
- DOM(document object model) get html elements, create html elements
- functions and anonumece functions
- if condition
- saving data to localStorage
- and a lot...

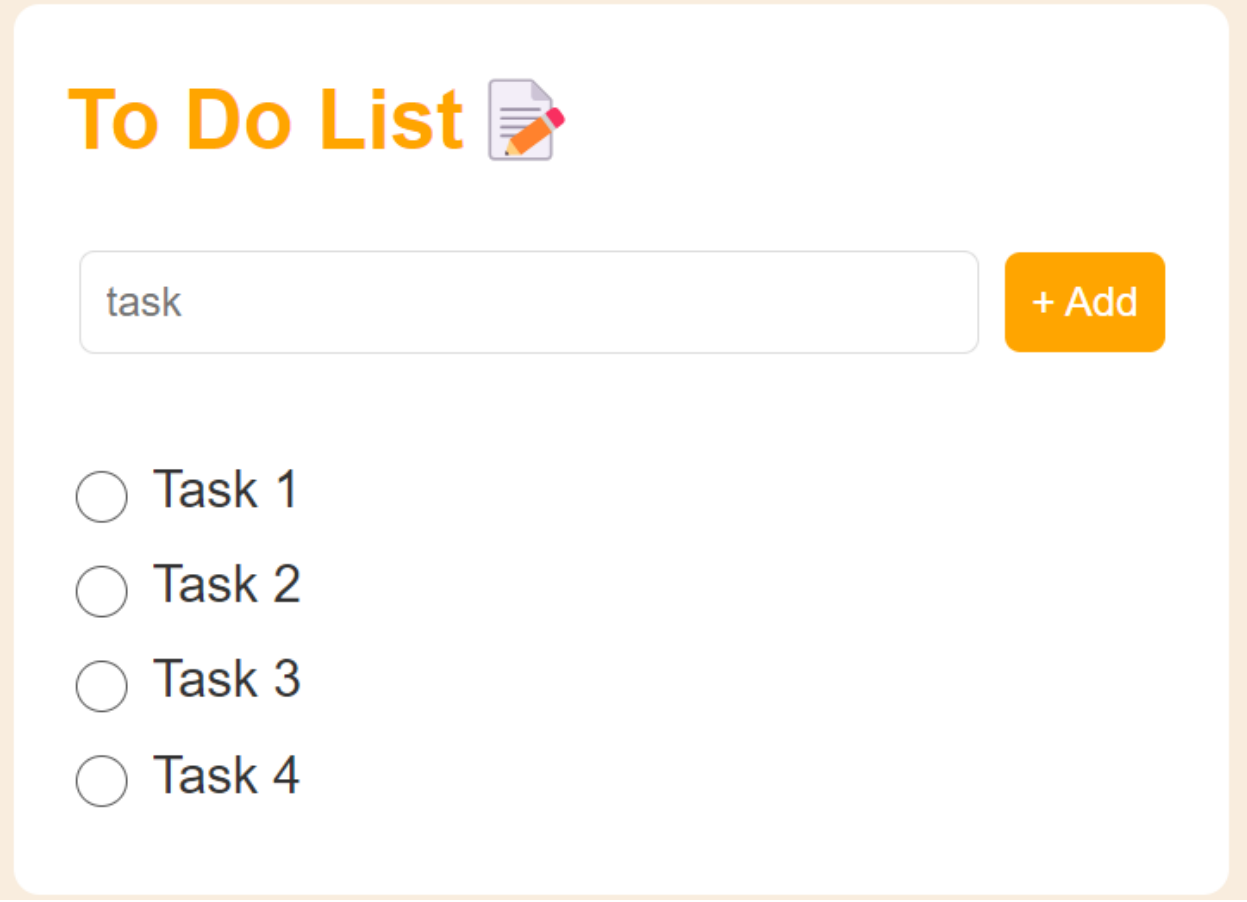
what do you know about javascript(conversation)?

JavaScript definition

JavaScript (JS) is a versatile, high-level language used mainly for web development. It enables interactive web pages, handles events, and manipulates the DOM. JavaScript runs both in browsers (client-side) and on servers (with Node.js). It supports asynchronous programming with callbacks, promises, and async/await, and powers frameworks like React, Angular, and Vue.

how this course work

we will build todo app to apply most of javascript concepts.



Requirement for this course

1. IDE
2. download files attached
3. don't hesitate asking questions

Download files

1. Open this link : https://github.com/Akira61/javascript_bootcamp
2. Click the green button which says “<> code”
3. Then click on “download ZIP”
4. After downloading unzip the folder

Explanation Process

☐Make sure HTML and CSS files are set up.

☐ View the website.

☐ Explain what we have in the HTML file

<lets code>

Chapter 1

Comments, declaring variables, and getting HTML elements.

-

Comment

1. One Line comment:

```
// get html elements
```

2. Multi-lines comment:

```
/*
 1. task one
 2. task two
*/
```

The Three ways to declare a variable

in JavaScript no data type specification needed.

1. Const, It is used if a variable's value is not going to change. Example:

```
const name = "Fahad Alshehri";
```

2. Let, it is used if a variable's value might change

Example:

```
let salary = 4000;
salary = 6500;
```

3. Var, is almost the same as let but in old JavaScript version (not recommended to use for scoping problems).

Example:

```
var salary = 4000;
salary = 6500;
```

Getting HTML elements

To miniplate HTML elements you first need to get the elements.

You can get and miniplates elements using the 'document' method declared in JavaScript.

You can get an element by its name, its class or its id.

There are many ways to get an element in JavaScript.

But we will discover the most popular ways.

1. QuerySelector,
   Get an element by its name.
   Write the name of the elements inside "quotations"

```
document.querySelector("input");
```

2. QuerySelector,

   Get an element by it class.

   Write a ‘.’ followed by the name of the class

```
document.querySelector(".new-task-holder");
```

3. QuerySelector,

   Get an element by its id.

   Write a ‘#’ followed by the name of the id

```
document.querySelector("#tasks");
```

   Or you can use getElementById:

```
document.getElementById("new-task");
```

4. QuerySelectorAll,

   To get multiple elements with the same specification

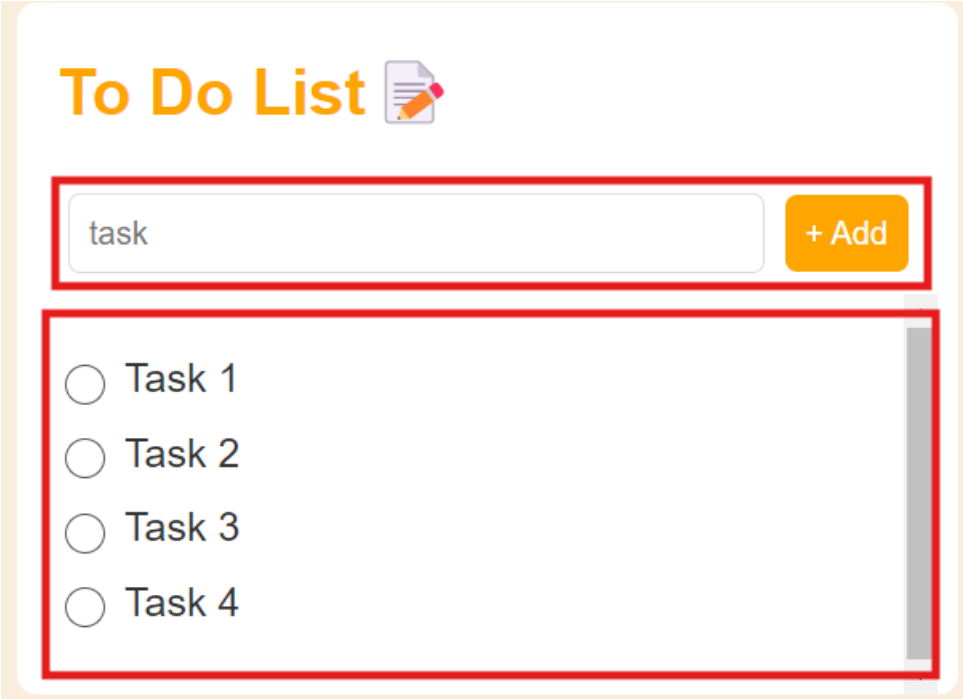
```
document.querySelectorAll("input[type=’checkbox’]");
```

   Here we are getting all inputs with ‘checkbox’ type.

Chapter 2

Adding Task

-



As showing in the photo, there are two sections in this application:

1. Adding a task
2. Showing tasks

In this chapter we will be focusing on Adding a Task.

Declaring HTML elements

To add a task, we need to know if a user typed a task and clicked the “+Add” button.

For that we need first to get the input element and the button element.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");
```

Button Event Listener

After that we need to add an Event Listener to “+Add” button.

The event Listener’s job is to watch the button if it got clicked.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {});
```

In the event listener, there is an “anonymous function”, it gets called automatically.

It used for one time uses (you can’t call the function anywhere else in the code).

There is also a parameter passed in the function called ‘e’ which has event properties.

Inside {} you write your block of code.

Event Listener methods

One of the event Listener methods is preventing the page from reloading for a better user experience.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  });
```

Prevent adding empty task

Before we add the task to the list, we need to make sure that the user typed something (You know... Don’t trust the user)

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
  }
```

```
});
```

Here we are checking if the input value is an empty string.

We can use ‘if’ statement as shown on the code.

To get what the user typed we use “input.value”.

Creating new elements in the document

```
<li class="task">
    <input type="checkbox" id="424234323">
    <label>Task 1</label>
</li>
```

As shown above, this is the way we create a task in HTML.

We want to do the same thing but in JavaScript.

To create an element, we use

```
document.createElement("elementName");
```

with the name of the element inside the “quotations”

Create the task holder (li)

First, we need to create the task holder which is the li and assign it properties.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
```

```
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {

    // create task holder
    const taskHolder = document.createElement("li");
    taskHolder.className = "task";
  }
});
```

To assign a class to an element we use “.className”.

Create the input checkbox (input)

Next, we need to create the input checkbox element.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {

    // create task holder
    const taskHolder = document.createElement("li");
    taskHolder.className = "task";
    // create checkbox for the task
    const checkbox = document.createElement("input");
    checkbox.type = "checkbox";
    const taskId = Date.now();
    checkbox.id = taskId;
  }
});
```

To set the type of input we use “.type”.

As we can see we are using the date as an id for the checkbox, there are many ways to make an element unique,

But the simplest way is to use the current date which includes milliseconds.

After that we assign the taskId to the checkbox using “.id”

Create the task title (label)

Now it's time for the task title. We will do the same thing we did previously.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {

     // create task holder
     const taskHolder = document.createElement("li");
     taskHolder.className = "task";
     // create checkbox for the task
     const checkbox = document.createElement("input");
     checkbox.type = "checkbox";
     const taskId = Date.now();
     checkbox.id = taskId;
     // set task title
     const taskTitle = document.createElement("label");
```

```
    taskTitle.innerHTML = newTaskInput.value;

  }
});
```

We are using “.innerHTML” to set label’s content.

Note that this method would activate HTML tags if you added them in the string.

Adding checkbox and label inside li

Finaly we should add the checkbox and the label inside the task holder which is li.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {

    // create task holder
    const taskHolder = document.createElement("li");
    taskHolder.className = "task";
    // create checkbox for the task
    const checkbox = document.createElement("input");
    checkbox.type = "checkbox";
    const taskId = Date.now();
    checkbox.id = taskId;
    // set task title
    const taskTitle = document.createElement("label");
    taskTitle.innerHTML = newTaskInput.value;
```

```
    taskHolder.appendChild(checkbox);
    taskHolder.appendChild(taskTitle);
  }
});
```

To add checkbox and label inside li we use ".appendChild".

Checkbox and label are considered as children of the li. Because they are inside li.

li is the parent of checkbox and label because these elements are inside li.

Clear input text

For a better user experience, let's clear the input text after creating the task.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {

    // create task holder
    const taskHolder = document.createElement("li");
    taskHolder.className = "task";
    // create checkbox for the task
    const checkbox = document.createElement("input");
    checkbox.type = "checkbox";
    const taskId = Date.now();
```

```
    checkbox.id = taskId;
    // set task title
    const taskTitle = document.createElement("label");
    taskTitle.innerHTML = newTaskInput.value;
    taskHolder.appendChild(checkbox);
    taskHolder.appendChild(taskTitle);
    // clear new task input
    newTaskInput.value = "";
  }
});
```

Creating a function

To make the code cleaner and more readable, let's create a function and call it inside the event listener.

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
    const task = newTask(newTaskInput.value);
    // clear new task input
    newTaskInput.value = "";
  }
});
```

```
function newTask(task) {
  // create task holder
  const taskHolder = document.createElement("li");
  taskHolder.className = "task";
  // create checkbox for the task
  const checkbox = document.createElement("input");
  checkbox.type = "checkbox";
  const taskId = Date.now();
  checkbox.id = taskId;
  // set task title
  const taskTitle = document.createElement("label");
  taskTitle.innerHTML = task;
  taskHolder.appendChild(checkbox);
  taskHolder.appendChild(taskTitle);
  return { task: taskHolder, taskTitle, taskId: checkbox.id };
}
```

We are creating a function with the name ‘newTask’ and passing a parameter, the taskmaster which is the task that the user wrote.

Next, we are returning an object with the info we need

Note that if you create a function, you can call it anywhere in the code

Chapter final code

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
    const task = newTask(newTaskInput.value);
```

```
    // clear new task input
    newTaskInput.value = "";
  }
});

function newTask(task) {
  // create task holder
  const taskHolder = document.createElement("li");
  taskHolder.className = "task";
  // create checkbox for the task
  const checkbox = document.createElement("input");
  checkbox.type = "checkbox";
  const taskId = Date.now();
  checkbox.id = taskId;
  // set task title
  const taskTitle = document.createElement("label");
  taskTitle.innerHTML = task;
  taskHolder.appendChild(checkbox);
  taskHolder.appendChild(taskTitle);
  return { task: taskHolder, taskTitle, taskId: checkbox.id };
}
```

Chapter 3

Storing Task

-

There are many ways to store data, such as data in database. But in our case, we will store the data on the browser.

To be able to store data in browser we will use “localStorage”.

The data stored in the browser's local storage will always stay there unless you delete it.

Introduction to Local Storage

Key	Value
name	fahad
age	20
college	Computer Science

Photo of local storage inside the browser

As we can see from the example above, the local storage uses an object data structure.

Object data structure

In JavaScript there is another way to store data which looks like:

```
const object = {
  name: "value",
  age: 20,
  alive: true
}
```

First, you declare a variable, then open object precis which looks like “{}”

The object contains two attributes:

1. Key
   It used to get the value you want. almost like index in an array
   The key should describe the data you want to store
2. Value
   Value which stores the actual value.

Note: the object can store different data types

Local Storage methods

This is how we write local storage in JavaScript:

```
localStorage.getItem("tasks");
```

There are several methods for local storage, but we will use only two which is:

1. Set item

```
localStorage.setItem("name", "Fahad Alshehri");
```

Set item is used to create new data, you pass to the function two arguments:

1. key
2. Value

value data type should always be string.

2. Get item

```
localStorage.getItem("name");
```

It used to get existing data; you pass one argument which is the key

Storing tasks

First, we need to create an array to dumb the tasks on it.

```
// store task
let storedTasks = [];
```

Now let's create an object to hold task info.

```
// store task
    let storedTasks = [];
    storedTasks.push({
      id: task.taskId,
      title: task.taskTitle.innerHTML,
    });
    console.log(storedTasks);
```

To add something to an array use the “.push” array method.

We have an object which contains two things:

1. Id
   We will store task id to identify the task.
   We can get the task id from the object returned in “newTask” function
2. Title
   We will store the actual task inside the title
   To get the content from task element we use “innerHTML”

To log anything inside the terminal we use

```
    console.log(storedTasks);
```

Before we check the task we made, let’s add our code inside button event listener:

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
    const task = newTask(newTaskInput.value);
    // clear new task input
    newTaskInput.value = "";
    // store task
    let storedTasks = [];
    storedTasks.push({
      id: task.taskId,
```

```
      title: task.taskTitle.innerHTML,
    });
    console.log(storedTasks);
  }
});

function newTask(task) {
  // create task holder
  const taskHolder = document.createElement("li");
  taskHolder.className = "task";
  // create checkbox for the task
  const checkbox = document.createElement("input");
  checkbox.type = "checkbox";
  const taskId = Date.now();
  checkbox.id = taskId;
  // set task title
  const taskTitle = document.createElement("label");
  taskTitle.innerHTML = task;
  taskHolder.appendChild(checkbox);
  taskHolder.appendChild(taskTitle);
  return { task: taskHolder, taskTitle, taskId: checkbox.id };
}
```

Bet if we reload the page our task will disappear.

Next, we will see how to prevent data from disappearing by storing the data in local storage

Storing tasks to local storage

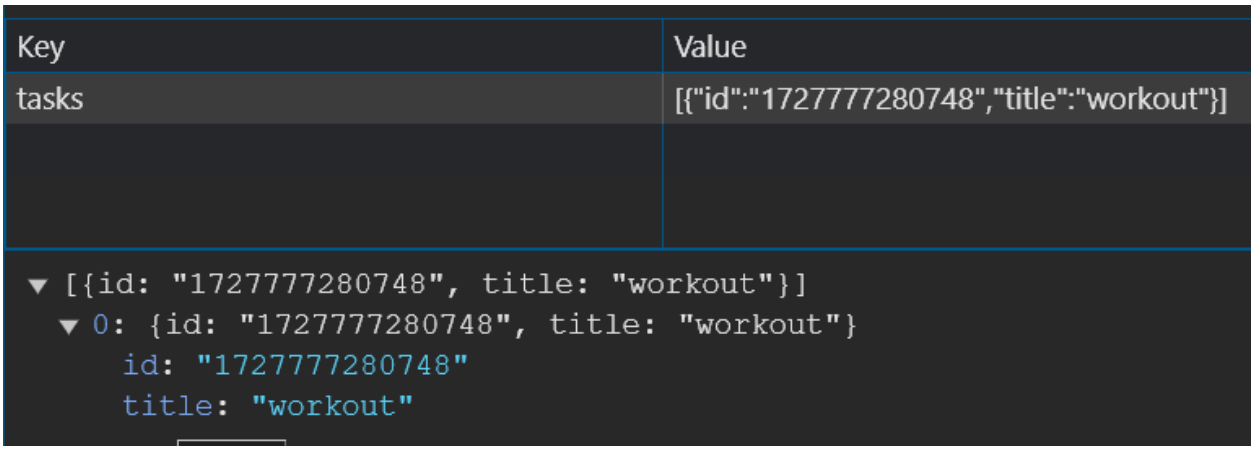
To store data in local storage we will use “setItem”

```
// store task
    let storedTasks = [];
    storedTasks.push({
      id: task.taskId,
      title: task.taskTitle.innerHTML,
```

```
    });
    console.log(storedTasks);
    localStorage.setItem("tasks", JSON.stringify(storedTasks));
```

Before we store an array of objects to local storage, we need first to convert the array to string using “JSON.stringify(storedTasks)”.

Have a look on local storage:



But there is a problom, if we reload the page and try to add a new task, it will delete all previous tasks and add the new one.

But why?

When we reload the page, the array of “stored Tasks” will be empty, because it is empty we will add the new task to an empty array and reset local Storage to be one task which is the new one

Solve saving issue when reload

Instead of adding the task inside an array. We will first get the existing tasks from local storage and add the new task to them.

```
// store task
    let storedTasks = localStorage.getItem("tasks");
    storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
    storedTasks.push({
      id: task.taskId,
      title: task.taskTitle.innerHTML,
    });
    console.log(storedTasks);
    localStorage.setItem("tasks", JSON.stringify(storedTasks));
```

First we are getting the existing tasks by “getItem”.

In the second line, we are checking whether there are tasks or the user just created one.

If there are tasks, we will convert the array of tasks from string to normal array using “JSON.parse(storedTasks)” .

If there are no tasks, we will create a new empty array.

Store tasks function

To make the code cleaner and easier to read, let's create a function and add storing data logic to it. we will call the function inside the event listener.

```
function storeTask(taskTitle, taskId) {
  let storedTasks = localStorage.getItem("tasks");
```

```
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  storedTasks.push({
    id: taskId,
    title: taskTitle,
  });
  // Save the updated data back to localStorage
  localStorage.setItem("tasks", JSON.stringify(storedTasks));
}
```

Here we are accepting two parameters, task title and task Id.

We are changing the stored Tasks object to the parameters.

Finaly let's call the function:

```
addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
    const task = newTask(newTaskInput.value);
    // clear new task input
    newTaskInput.value = "";
    // store task
    storeTask(task.taskTitle.textContent, task.taskId);
  }
});
```

Chapter final code

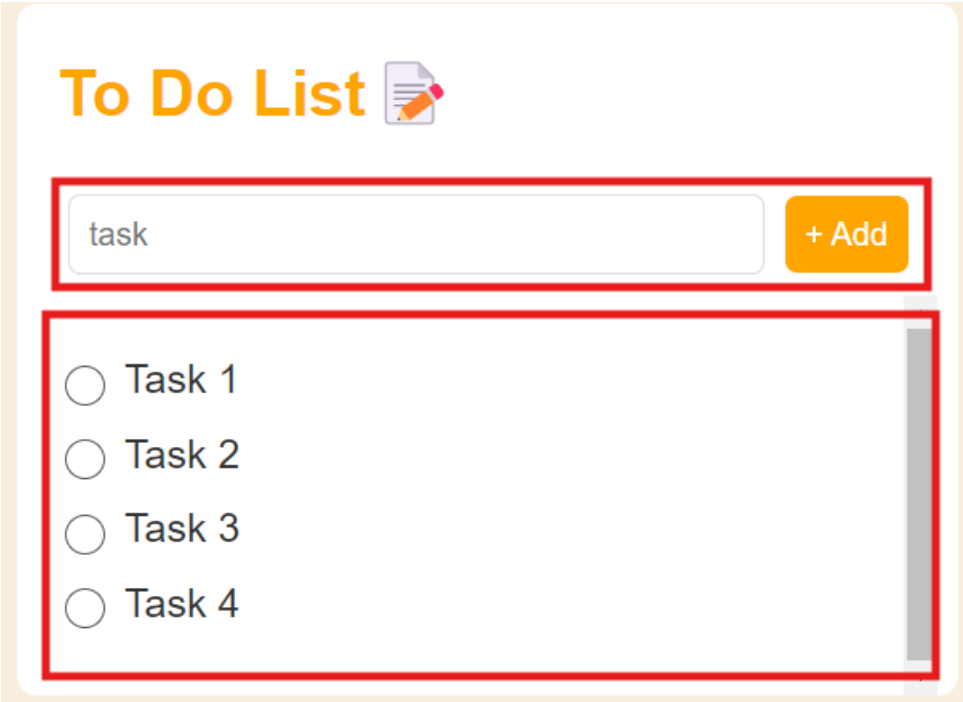
```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
```

```
    // create task
    const task = newTask(newTaskInput.value);
    // clear new task input
    newTaskInput.value = "";
    // store task
    storeTask(task.taskTitle.textContent, task.taskId);
  }
});

function newTask(task) {
  // create task holder
  const taskHolder = document.createElement("li");
  taskHolder.className = "task";
  // create checkbox for the task
  const checkbox = document.createElement("input");
  checkbox.type = "checkbox";
  const taskId = Date.now();
  checkbox.id = taskId;
  // set task title
  const taskTitle = document.createElement("label");
  taskTitle.innerHTML = task;
  taskHolder.appendChild(checkbox);
  taskHolder.appendChild(taskTitle);
  return { task: taskHolder, taskTitle, taskId: checkbox.id };
}

function storeTask(taskTitle, taskId) {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  storedTasks.push({
    id: taskId,
    title: taskTitle,
  });
  // Save the updated data back to localStorage
  localStorage.setItem("tasks", JSON.stringify(storedTasks));
}
```

Chapter 4

Show tasks

-



Now let’s focus on the second part of the program which shows the tasks.

We will create a function called showTask.

Show task function

```
showTask();
function showTask() {
  // show the tasks
}
```

We are creating the function and calling it, so when the page reloads it will show tasks.

Now let's get tasks from local storage:

```
function showTask() {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
}
```

Show tasks inside html

To show tasks inside html we first need to get the tasks holder in html which is an ‘ul’ element with id of “tasks”.

On the top of the code, we add this line:

```
const tasks = document.querySelector("#tasks");
```

After we get the tasks holder, let’s remove task template inside tasks HTML:

```
function showTask() {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  // clear tasks templete
  tasks.innerHTML = "";
}
```

Set tasks content to empty string.

Next, we will use for loop for each task and add it to the tasks

For loop

In JavaScript this is how you create a for loop:

```
for (let i = 0; i < storedTasks.length; i++) {
    // do something
 }
```

Same as Java.

To get array length we use “.length”.

Create the task

```html
<li class="task">
      <input type="checkbox" id="424234323">
      <label>Task 1</label>
</li>
```

Task element in HTML

We will introduce a new way of declaring HTML elements:

```
function showTask() {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  // clear tasks templete
  tasks.innerHTML = "";
  // add stored tasks
  for (let i = 0; i < storedTasks.length; i++) {
    tasks.innerHTML += `
        <li class="task">
            <input type="checkbox" id="${storedTasks[i].id}">
            <label for="${storedTasks[i].id}">${storedTasks[i].title}</label>
        </li>
        `;
  }
}
```

I will loop throw each task in local Storage and create html element for that task and append it to tasks holder(ul).

We can use “.innerHTML” and write html tags in the string.

Note that I’m using ‘+=’ for appending task.

I’m using what is known as backtick (`).

It allows me to write multi-line string,

Also, it gives me access to variables using “${variable}”.

Show tasks when adding one

Let's add the showTask function to the button event listener so that if we added a new task, we see it:

```
addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
    const task = newTask(newTaskInput.value);
    // clear new task input
    newTaskInput.value = "";
    // store task
    storeTask(task.taskTitle.textContent, task.taskId);
    // show all tasks
    showTask();
  }
});
```

Chapter final code

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");
const tasks = document.querySelector("#tasks");

showTask(); // show tasks on page reload

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
    const task = newTask(newTaskInput.value);
    // clear new task input
    newTaskInput.value = "";
    // store task
    storeTask(task.taskTitle.textContent, task.taskId);
    // show all tasks
    showTask();
  }
});


function showTask() {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  // clear tasks templete
  tasks.innerHTML = "";
  // add stored tasks
  for (let i = 0; i < storedTasks.length; i++) {
    tasks.innerHTML += `
        <li class="task">
            <input type="checkbox" id="${storedTasks[i].id}">
            <label for="${storedTasks[i].id}">${storedTasks[i].title}</label>
        </li>
        `;
  }
}

function newTask(task) {
```

```
  // create task holder
  const taskHolder = document.createElement("li");
  taskHolder.className = "task";
  // create checkbox for the task
  const checkbox = document.createElement("input");
  checkbox.type = "checkbox";
  const taskId = Date.now();
  checkbox.id = taskId;
  // set task title
  const taskTitle = document.createElement("label");
  taskTitle.innerHTML = task;
  taskHolder.appendChild(checkbox);
  taskHolder.appendChild(taskTitle);
  return { task: taskHolder, taskTitle, taskId: checkbox.id };
}

function storeTask(taskTitle, taskId) {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  storedTasks.push({
    id: taskId,
    title: taskTitle,
  });
  // Save the updated data back to localStorage
  localStorage.setItem("tasks", JSON.stringify(storedTasks));
}
```

Final Chapter

Remove task

-

Now we can add a task, save the task, and show the task. It is time to delete the task when completed.

First, we need to add an event listener for all input checkboxes to see if it got clicked

Tasks event listener

We will add an event listener for the tasks holder and check if a checkbox inside the tasks holder got clicked.

```
// if checkbox clicked remove task
tasks.addEventListener("click", (e) => {
  // check if element is checkbox
  if (e.target.tagName === "INPUT") {
    // delete element from localStorage
  }
});
```

There is a method inside the event listener (e) that can get the tag name of the element clicked.

Here we are checking if the input tag is the one clicked inside the tasks holder.

If it is, we will call a function that we will write about later to remove the tasks.

```
// if checkbox clicked remove task
tasks.addEventListener("click", (e) => {
  // check if element is checkbox
  if (e.target.tagName === "INPUT") {
    // delete element from localStorage
    removeTask(e.target.id);
    // remove element from page
    e.target.parentElement.remove();
  }
});
```

To get the element's id we use "e.target.id".

Here we are passing an argument to the "removeTask" function which is the checkbox id.

After that we are removing this element from tasks holder using "e.target.parentElement.remove()"

The tasks considered as parent for the task.

So, we are removing from tasks a task.

Remove task function

As we saw above, we called a function called “removeTask”.

This function contains:

```
function removeTask(taskId) {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = JSON.parse(storedTasks);
  console.log(storedTasks);
  const filterTasks = storedTasks.filter((task) => task.id !== taskId);
  localStorage.setItem("tasks", JSON.stringify(filterTasks));
}
```

We are taking tasks from local storage and parsing it to an array.

Here we have a new array method which is “filter”.

Filter

Filter takes an argument which is an array element and will return only the elements that match the condition.

In our case the condition is:

```
task.id !== taskId
```

Which means, give me the task that it is id does not equal id passed in the function

Finaly we are taking the filtered tasks and storing it again.

final code

```
const newTaskInput = document.getElementById("new-task");
const addTaskButton = document.getElementById("new-task-btn");
const tasks = document.querySelector("#tasks");

showTask(); // show tasks on page reload

addTaskButton.addEventListener("click", (e) => {
  e.preventDefault();
  // prevent adding task if input empty
  if (newTaskInput.value !== "") {
    // create task
    const task = newTask(newTaskInput.value);
    // clear new task input
    newTaskInput.value = "";
    // store task
    storeTask(task.taskTitle.textContent, task.taskId);
    // show all tasks
    showTask();
  }
});

// if checkbox clicked remove task
tasks.addEventListener("click", (e) => {
  // check if element is checkbox
  if (e.target.tagName === "INPUT") {
    // delete element from localStorage
    removeTask(e.target.id);
    // remove element from page
    e.target.parentElement.remove();
  }
```

```
});


function showTask() {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  // clear tasks templete
  tasks.innerHTML = "";
  // add stored tasks
  for (let i = 0; i < storedTasks.length; i++) {
    tasks.innerHTML += `
        <li class="task">
            <input type="checkbox" id="${storedTasks[i].id}">
            <label for="${storedTasks[i].id}">${storedTasks[i].title}</label>
        </li>
        `;
  }
}

function newTask(task) {
  // create task holder
  const taskHolder = document.createElement("li");
  taskHolder.className = "task";
  // create checkbox for the task
  const checkbox = document.createElement("input");
  checkbox.type = "checkbox";
  const taskId = Date.now();
  checkbox.id = taskId;
  // set task title
  const taskTitle = document.createElement("label");
  taskTitle.innerHTML = task;
  taskHolder.appendChild(checkbox);
  taskHolder.appendChild(taskTitle);
  return { task: taskHolder, taskTitle, taskId: checkbox.id };
}

function storeTask(taskTitle, taskId) {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = storedTasks ? JSON.parse(storedTasks) : [];
  storedTasks.push({
    id: taskId,
    title: taskTitle,
  });
  // Save the updated data back to localStorage
```

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

function removeTask(taskId) {
  let storedTasks = localStorage.getItem("tasks");
  storedTasks = JSON.parse(storedTasks);
  console.log(storedTasks);
  const filterTasks = storedTasks.filter((task) => task.id !== taskId);
  localStorage.setItem("tasks", JSON.stringify(filterTasks));
}
```

End :(