

**FREE**

# JavaScript

**CREATE YOUR OWN  
JAVASCRIPT  
LIBRARY**

Sagar Kumar

# INDEX

<b>1) Setting up your project</b>	<b>1</b>
<b>2) Creating your first JavaScript Library</b>	<b>4</b>

# Chapter 1

## Setting up your project

### INTRODUCTION:

Hello, dear reader! Are you still using jQuery for your projects? If so, it's time to rethink that approach. Instead of relying on jQuery, why not take a step forward and create your very own JavaScript library? In this chapter, I'll guide you through the process of building a lightweight, fast, and easy-to-use library.

But before we begin, let's take a moment to understand why creating your own library is a great idea.

### Why Not jQuery?

- jQuery has been a popular library for simplifying DOM manipulation and AJAX calls. However, modern JavaScript has evolved, and many of jQuery's features are now natively available in the language itself.
- Creating your own library not only helps you learn but also gives you complete control over its functionality.

### Advantages of Our Library:

- Lightweight and super fast.
- No need to memorize jQuery's syntax.
- Simplifies the code-writing process, just like jQuery.
- Enables direct use of JavaScript properties and methods, something jQuery abstracts away.

```
//With jQuery
$('#myDiv').style.color = "red"; // Error: undefined

//With Our Library
select('#myDiv').style.color = "red" // Works
```

Looks exciting, right? Let's get Started

## Step 1: Setting Up Your Project

First, let's create a basic project structure. Follow these steps:

### 1. Create a Folder:

- Create a new folder anywhere on your system. You can name it something like my-library-project.


### 2. Open the Folder in Visual Studio Code:

- If you're using Visual Studio Code (VS Code), open the folder by navigating to File > Open Folder or simply drag and drop the folder into the editor.

### 3. Add the Required Files: Inside the folder, create the following files:

- index.html – This will be the HTML file for testing and demonstration.
- style.css – This file will contain the styling for our project.
- app.js – This will hold our JavaScript code for testing.
- mylibrary.js – This is where we will write the code for our custom library.

Your project structure should look like this:



```
my-library-project/  
├─ index.html  
├─ style.css  
├─ app.js  
└─ mylibrary.js
```

## Step 2: Understanding How jQuery Works

Before building our library, let's briefly see how jQuery simplifies things. In this example, we'll create some basic HTML elements and apply styles to them.

## 1. Create a Container Element:

Open the index.html file and add the following code:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My JavaScript Library</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <div class="box"></div>
    <div class="box"></div>
    <div class="box"></div>
  </div>
  <script src="mylibrary.js"></script>
  <script src="app.js"></script>
</body>
</html>

```

- The `<div class="container">` contains three child elements with the class `box`. These will be styled as basic boxes later.
- Don't forget to link `style.css` and the two JavaScript files (`mylibrary.js` and `app.js`) at the bottom of the body section.

## 2. Apply Basic Styles:

Open the style.css file and add the following styles:

```

body {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.container {
  display: flex;
  gap: 10px;
}

.box {
  width: 100px;
  height: 100px;
  background-color: #3498db;
  border-radius: 5px;
}

```

This will center the boxes on the screen and apply a simple blue color to them.

# Chapter 2

## Creating your first JavaScript Library

### INTRODUCTION:

In this chapter, we will create a custom JavaScript library that can select elements, apply styles, and handle events—just like jQuery. However, unlike jQuery, this library will allow you to use native JavaScript properties and methods, giving you more flexibility and control.

Let's dive in!

### The `select()` Function: An Overview

Our library revolves around a single function, `select()`, which simplifies common tasks such as selecting DOM elements, applying styles, and adding event listeners. Here's the complete function:

```
function select(selector) {
  let element = document.querySelectorAll(selector);

  for (let i = 0; i < element.length; i++) {
    element[i].makeUp = function (styles) {
      if (styles) {
        this.style.cssText = styles;
      } else {
        return this.style.cssText;
      }
    };

    element[i].on = function (event, callback) {
      this.addEventListener(event, callback);
    };
  }

  if (element.length == 1) {
    return element[0];
  } else {
    return element;
  }
}
```

## Step 1: Selecting Elements

The first step is to select the elements that match a given CSS selector. This is done using the `document.querySelectorAll()` method:

```
let element = document.querySelectorAll(selector);
```

- **Input:** A string selector, such as `.class` (for class), `#id` (for ID), or `tagname` (for HTML tags).
- **Output:** A `NodeList` containing all matching elements in the DOM.

## Step 2: Adding Methods to Each Element

Once we have the `NodeList`, we loop through each element and add custom methods to extend its functionality.

```
for (let i = 0; i < element.length; i++) {  
  element[i].makeUp = function (styles) {  
    if (styles) {  
      this.style.cssText = styles;  
    } else {  
      return this.style.cssText;  
    }  
  };  
  
  element[i].on = function (event, callback) {  
    this.addEventListener(event, callback);  
  };  
}
```

## Method 1: The `makeUp()` Method

The `makeUp()` method allows you to apply inline styles to an element or retrieve its current styles.

### Input:

- A string `styles` containing valid CSS rules (e.g., `"color: red; font-size: 20px;"`).

### Output:

- If a `styles` argument is provided, the method applies the styles to the element.
- If no argument is provided, it returns the current inline styles of the element

Example Code:

```
let box = select('.box')[0];
box.makeUp("color: blue; background-color: yellow;");

// Retrieve current styles
console.log(box.makeUp()); // Output: "color: blue; background-color: yellow;"
```

## Method 2: The on() Method

The on() method simplifies adding event listeners to elements.

**Input:**

- event: A string specifying the event type (e.g., "click", "mouseover").
- callback: A function to execute when the event occurs.

**Output:** None. It attaches the event listener to the element.

Example Code:

```
let box = select('.box')[0];
box.on("click", () => {
  alert("Box clicked!");
});
```

## Step 3: Handling Single vs. Multiple Elements

The select() function determines whether the selector matches a single element or multiple elements. If only one element matches, it returns the element directly. Otherwise, it returns the entire NodeList.

```
if (element.length == 1) {
  return element[0];
} else {
  return element;
}
```

Example Code:

```
let singleBox = select('#unique-box'); // Returns a single element
let allBoxes = select('.box'); // Returns a NodeList of elements
```



## Step 4: Using the Library

Now that we've built the library, let's see how it works in practice.

### 1. HTML Setup:

```

<div class="box">Box 1</div>
<div class="box">Box 2</div>
<div class="box">Box 3</div>

```

### 2. JavaScript Example:

```

// Select all boxes
let boxes = select('.box');

// Apply styles to all boxes
for (let box of boxes) {
  box.makeUp("color: white; background-color: green; padding: 10px;");
}

// Add a click event to the first box
let firstBox = select('.box')[0];
firstBox.on("click", () => {
  alert("You clicked the first box!");
});

```

## Why Our Library Is Better Than jQuery

Here are some reasons why this custom library stands out:

1. Lightweight: Unlike jQuery, our library has no external dependencies and focuses on essential features.
2. Direct JavaScript Methods: You can still use all native JavaScript methods and properties on the selected elements.
3. Customizable: You can easily add more methods to extend the functionality.
4. Learning Opportunity: By building your own library, you gain a deeper understanding of JavaScript and DOM manipulation.

<https://code.jquery.com/jquery-3.7.1.min.js>

Click on the link above and take a look at how large the jQuery file is, which is not necessary for us. Instead, create your own JavaScript library.

## Enhancing the Library (Optional)

Want to expand the library further? Here are some ideas:

- Add a method for toggling classes, such as `toggleClass(className)`.
  - Add a method for AJAX calls, similar to `$.ajax()` in jQuery.
  - Implement animations, such as fading in or out elements.
-

**Thank You**