JavaScript

101

INTRODUCTION

Title: ***JavaScript 101***
Compilation : ***Reinhard Behrens***
Edition: ***1st Edition***

Cover Design: ***ChatGPT Image Generation Tool***

Disclaimer
This book contains references to various tools, technologies, and programming practices. Certain sections have incorporated code examples generated with the assistance of ChatGPT, an AI language model developed by OpenAI. While efforts have been made to ensure the accuracy and quality of the content, the publisher and author(s) are not liable for any errors or omissions.

ChatGPT Disclaimer for Code Generation
The code snippets and programming solutions presented in this book have been generated with the help of ChatGPT. ChatGPT is an AI model that can assist with coding tasks, but its responses may not always be perfectly optimized or error-free. Readers are encouraged to test, review, and modify the code examples according to their specific use cases. The compiler of this book has tested each exercise for proper functioning.

Thank you JESUS for making the tools and time available for me to complete this booklet and to go through all the exercises. Without God I would not have been able to have the resources and availability to complete this book. All glory to the Son of GOD.

Contents

Introduction

This booklet is designed for people that is busy learning JavaScript and require an exercise book to learn JavaScript. When doing Video tutorials or courses, they often forget to include multiple different exercises and practical real world examples. JavaScript is both a functional programming language as well as a OOP style programming language. This book is designed for someone that is learning JavaScript from scratch, and would like to go through these exercises to solidify your understanding of JavaScript.

At the time of generating this booklet, I was, like you (assuming you are new to JavaScript), a newbie to the JavaScript language.

To get the most value from this booklet don't just copy and paste the code, but type out the entire exercise to learn the language and understand how it is used in the real world. Use ChatGPT to generate your own sample projects once you have finished the samples of this book. This will improve your learning curve on understanding complex code. So lets learn to code JavaScript together, and overcome the fear for becoming a great JavaScript developer.

Why JavaScript?

JavaScript is used by companies such as Netflix, PayPal, LinkedIn, Walmart, Uber, eBay, and Yahoo. They use JavaScript because you can develop for both front-end web, develop Mobile UI's using React Native and also develop back-end code such NodeJS backend software applications. With NodeJS latest additions of Worker Threads, NodeJS can start competing with Java and other languages which has for a long time been used for corporate backends.

For example, for FinTech moving to NodeJS has the following advantages:

- 50% faster processing times
- 30% lower server costs due to lightweight architecture
- 82% prefers Node.js due to quick response times

(Ref : https://webcluesinfo.medium.com/node-js-in-fintech-revolutionizing-financial-services-2aa53a792b2d)

The language has multiple mature UI frameworks such as React, Angular, Vue.js, and Svelte. JavaScript is also used in web games programming and 3D visualization.

So join the JavaScript revolution and help 16.5 million other JavaScript developers serving clients and/or build software development businesses.

JavaScript Basics – 1 to 25 Exercises with Answers

---

1. Declare and Log a Variable

Task: Declare a variable name with your name and log it.

Solution:

```
let name = "Alice";

console.log(name);
```

---

2. Determine Data Types

Task: Identify the data types of the following values: 42, "hello", true, null, undefined.

Solution:

```
console.log(typeof 42);        // number

console.log(typeof "hello");    // string

console.log(typeof true);       // boolean

console.log(typeof null);       // object (quirk in JS)

console.log(typeof undefined);  // undefined
```

---

3. Use Arithmetic Operators

Task: Add, subtract, multiply, and divide two numbers.

Solution:

```
let a = 10, b = 5;

console.log(a + b); // 15

console.log(a - b); // 5

console.log(a * b); // 50

console.log(a / b); // 2
```

---

4. Use Comparison Operators

Task: Compare two numbers using >, <, ==, and ===.

Solution:

```
let x = 7, y = "7";

console.log(x > 5);     // true

console.log(x < 10);    // true

console.log(x == y);    // true (loose equality)

console.log(x === y);   // false (strict equality)
```

---

5. Write an If-Else Condition

Task: Check if a number is positive, negative, or zero.

Solution:

```
let num = -3;

if (num > 0) {

   console.log("Positive");

} else if (num < 0) {

   console.log("Negative");

} else {

   console.log("Zero");

}
```

---

6. Use a Switch Statement

Task: Print the day of the week based on number (1-7).

Solution:

```
let day = 3;

switch (day) {

   case 1: console.log("Monday"); break;
```

```
  case 2: console.log("Tuesday"); break;
  case 3: console.log("Wednesday"); break;
  case 4: console.log("Thursday"); break;
  case 5: console.log("Friday"); break;
  case 6: console.log("Saturday"); break;
  case 7: console.log("Sunday"); break;
  default: console.log("Invalid day");
}
```

---

7. Use a For Loop

Task: Print numbers from 1 to 5.

Solution:

```
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

---

8. Use a While Loop

Task: Print numbers from 5 to 1.

Solution:

```
let i = 5;
while (i >= 1) {
  console.log(i);
  i--;
}
```

---

9. Write a Function

Task: Create a function that returns the square of a number.

Solution:

```
function square(n) {
  return n * n;
}
console.log(square(4)); // 16
```

---

10. Function with Default Parameter

Task: Create a function with a default parameter.

Solution:

```
function greet(name = "Guest") {
  console.log("Hello, " + name);
}
greet("Sam");  // Hello, Sam
greet();      // Hello, Guest
```

---

11. Array Basics

Task: Create an array of fruits and print the second one.

Solution:

```
let fruits = ["apple", "banana", "mango"];
console.log(fruits[1]); // banana
```

---

12. Add and Remove from Arrays

Task: Add an element to end, remove from start.

Solution:

```
let colors = ["red", "blue"];

colors.push("green");

colors.shift();

console.log(colors); // ["blue", "green"]
```

---

13. Object Basics

Task: Create an object for a person with name and age.

Solution:

```
let person = {
  name: "John",
  age: 30
};

console.log(person.name); // John
```

---

14. Access Object Properties

Task: Print all keys and values of an object.

Solution:

```
let car = { brand: "Toyota", year: 2020 };

for (let key in car) {
  console.log(key + ": " + car[key]);
}
```

---

15. String Length

Task: Get the length of a string.

Solution:

```
let message = "Hello World";

console.log(message.length); // 11
```

---

16. Convert to Uppercase

Task: Convert a string to uppercase.

Solution:

```
let city = "london";

console.log(city.toUpperCase()); // LONDON
```

---

17. Find a Character

Task: Get the character at index 3 of a string.

Solution:

```
let word = "JavaScript";

console.log(word.charAt(3)); // a

or

console.log(word.at(3)); //a
```

---

18. Parse String to Number

Task: Convert "42" to a number.

Solution:

```
let str = "42";

let num = Number(str);

console.log(num + 1); // 43
```

---

19. Round a Decimal Number

Task: Round 4.7 down and up.

Solution:

```
console.log(Math.floor(4.7)); // 4

console.log(Math.ceil(4.7));  // 5
```

---

20. Generate a Random Number (0–9)

Task: Generate a random number between 0-9 and print it the console.

Solution:

```
let rand = Math.floor(Math.random() * 10);

console.log(rand);
```

---

21. Use Logical Operators

Task: Check if age is between 18 and 65. Thus declare a variable and check if the variable range is between 18 and 65.

Solution:

```
let age = 25;

if (age >= 18 && age <= 65) {

   console.log("Valid age");

} else {

   console.log("Invalid age");

}
```

---

22. Ternary Operator

Task: Use a ternary to check if number is even. Try and use this function whenever possible, it creates clean code.

Solution:

```
let n = 4;
```

```
console.log(n % 2 === 0 ? "Even" : "Odd");
```

or

```
// You can create a function that does the same calculation
function checkNumberEven(number)
{
        return (number % 2 === 0) ? true : false;
}
```

---

23. Nested If Conditions

Task: Check if number is positive and divisible by 5.

Solution:

```
if(numberGreaterThanFive > 0)
{
  if(numberGreaterThanFive % 5 === 0)
  {
    console.log("Number devisable by 5 and is positive");
  }
  else
  {
    console.log("Number positive and not devisable by 5");
  }
}
else if(numberGreaterThanFive < 0)
{
  if(numberGreaterThanFive % 5 === 0)
```

```
  {
    console.log("Number devisable by 5 and is negative");
  }
  else
  {
    console.log("Number negative and not devisable by 5");
  }
}
```

---

24. Check for Element in Array

Task: Check if "apple" is in the array. You have to declare your own array for this.

Solution:

```
let items = ["banana", "apple", "orange"];

console.log(items.includes("apple")); // true
```

---

25. Basic Error Handling

Task: Use try-catch to handle error.

Solution:

```
try {
  let result = someUndefinedFunction();
} catch (error) {
  console.log("Caught an error:", error.message);
}
```

Note: The undefined function is like that on purpose, you can cause an error in other ways too, like using a null value or similar.

Here's the full **JavaScript Intermediate: 25 Exercises with Answers**, focusing on DOM manipulation, ES6 features, events, array/object methods, and basic async programming.

JavaScript Intermediate – 26-50 Exercises with Answers

---

26. Add an Event Listener

Task: Add a click event to a button that logs "Clicked!".

(Note: You can use the onclick event of the button for learning that way of doing to, see below)

HTML:

```
<button id="myBtn">Click Me</button>
```

JS:

```
document.getElementById("myBtn").addEventListener("click", () => {
  console.log("Clicked!");
});
```

or

HTML:

```
<button onclick="clickme();">Click me</button>
```

JS:

```
function clickme()
{
  console.log("Clicked!");
}
```

---

27. Change Element Text with JavaScript

Task: Create a HTML file with JavaScript to change the element text with new text.

HTML:

```
<p id="text">Old text</p>

<button onclick="changeText()">Change</button>
```

JS:

```
function changeText() {
  document.getElementById("text").textContent = "New text";
}
```

---

28. Use querySelector

Task: Change background color of an element using querySelector.

```
document.querySelector("div.box").style.backgroundColor = "yellow";
```

---

29. Toggle a CSS Class

Task: Toggle a class active on a button click.

HTML:

```
<button id="toggleBtn">Toggle Me</button>
```

CSS:

```
.active {
  background-color: green;
  color: white;
}
```

JS:

```
let btn = document.getElementById("toggleBtn");

btn.addEventListener("click", () => {
  btn.classList.toggle("active");
```

```
});
```

---

30. Loop through an Array with forEach

Task: Loop through an array, number or text array

```
let forEachArray = ["Apple","Banana","Pears"];

forEachArray.forEach((item, index)=>{

   console.log(`Index -> ${index} : Value -> ${item}`);

});
```

---

31. Filter Even Numbers

Task: Filter an array from 1-6 and print only the even numbers.

```
let numbers = [1, 2, 3, 4, 5, 6];

let evens = numbers.filter(n => n % 2 === 0);

console.log(evens); // [2, 4, 6]
```

---

32. Use .map() to Transform an Array

Task: Use the JavaScript to generate the .map() function to transform the array of text to UPPERCASE.

```
let toUppercaseArray = ['Apple', 'Pear', 'Bannana'];

let upperCaseArray = toUppercaseArray.map((n) => n.toUpperCase());

console.log(upperCaseArray);
```

---

33. Find an Element in an Array

```
let users = [{ id: 1 }, { id: 2 }, { id: 3 }];

let user = users.find(u => u.id === 2);

console.log(user); // { id: 2 }
```

---

34. Destructure an Object

```
let person = { name: "John", age: 30 };

let { name, age } = person;

console.log(name, age); // John 30
```

Note: This is a very important operation to remember as this is used a lot when working with third-party libraries, so make a note. You can always come back to this exercise in the future if you need to remember 😊

---

35. Destructure an Array

```
let arr = [10, 20, 30];

let [a, , c] = arr;

console.log(a, c); // 10 30
```

---

36. Use the Spread Operator

```
let nums1 = [1, 2];

let nums2 = [3, 4];

let all = [...nums1, ...nums2];

console.log(all); // [1, 2, 3, 4]
```

Note: In the implementation samples provided with this booklet, there is another option on how to use spread to bind two arrays with each other, so be sure to learn both.

---

37. Merge Two Objects with Spread

```
let o1 = { a: 1 };

let o2 = { b: 2 };

let merged = { ...o1, ...o2 };

console.log(merged); // { a: 1, b: 2 }
```

---

38. Template Literals

```
let name = "Sarah";
```

```
let greeting = `Hello, ${name}!`;

console.log(greeting); // Hello, Sarah!
```

---

39. Arrow Function

```
const add = (x, y) => x + y;

console.log(add(5, 3)); // 8
```

---

40. Default Parameters

```
function sayHi(name = "Guest") {
   console.log("Hi, " + name);
}

sayHi();       // Hi, Guest

sayHi("Leo");   // Hi, Leo
```

---

41. Rest Parameters

```
function sum(...nums) {
   return nums.reduce((a, b) => a + b);
}

console.log(sum(1, 2, 3)); // 6
```

---

42. Object.keys(), Object.values()

```
let user = { name: "Tom", age: 25 };

console.log(Object.keys(user));   // ["name", "age"]

console.log(Object.values(user)); // ["Tom", 25]
```

---

43. Create an Element and Append

Note: You can put this into a function if you like, this will make your code cleaner, try and compile this in a function instead of just typing this into the JavaScript file.

JS:

```
let p = document.createElement("p");

p.textContent = "New Paragraph";

document.body.appendChild(p);
```

---

44. Remove an Element from the DOM

```
let el = document.getElementById("toRemove");

el.remove();
```

---

45. Use setTimeout

```
setTimeout(() => {
  console.log("Executed after 1 second");
}, 1000);
```

---

46. Use setInterval and clearInterval

```
let count = 0;

let intervalId = setInterval(() => {
  count++;
  console.log(count);
  if (count === 5) clearInterval(intervalId);
}, 1000);
```

---

47. Fetch API – Basic GET Request

Note: This exercise uses Promise that will be explained later. For now just learn the syntax. Gemini was credit for this comprehensive answer on the fetch API and correct implementation.

JS:

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Data fetched successfully:', data);
  })
  .catch(error => {
    console.error('There was a problem fetching the data:', error);
  });
```

---

48. Async/Await Fetch

This exercise is for asynchronous function calls, don't be intimidated but rather just do the exercise and see where **async** is used and how **await** if used to wait for the response to process.

JS:

```
async function loadData() {
  const url = "https://jsonplaceholder.typicode.com/users/1";
  try {
    const response = await fetch(url);
```

```
    if (!response.ok) {
      throw new Error(`Response status: ${response.status}`);
    }

    const json = await response.json();
    console.log(`Await Example : \n${JSON.stringify(json, null, 2)}\n`);
  } catch (error) {
    console.error(error.message);
  }
 }
}
loadData();
```

---

49. Handle Errors in Fetch

Task: In this section you will purposefully insert a incorrect URL in an asynchronous call using fetch.

JS:

```
async function getUserData(){
  try{
    let response = await fetch("badurl");
    let httpResStatus = await response.ok;
    if(!httpResStatus)
    {
      console.log(`HTTP Status Code != 200 / Returned ${response.status}` );
    }
  }catch(error){
    console.error(error);
  }
}
getUserData();
```

---

50. DOMContentLoaded Event

Task: Create a function to determine if the DOM has loaded, the function should return true once its ready. Hint: You can use JavaScript Promises.

Note: Even if this is hard, try memorizing and checking why the use of a Promise is ideal for checking if the Document Object Model is loaded. Learn promises.

JS:

```
function isDOMReady()
{
  return new Promise((resolve => {
      if(document.readyState == "loading"){
```

```
            document.addEventListener("DOMContentLoaded", () => resolve(true) );
        }
        else{
            resolve = true;
        }
    }));
}
isDOMReady().then(()=> {
    console.log("DOM Ready");
});
```

JavaScript Advanced – 51 to 75 Exercises with Answers

---

51. Closure Example

Task: Create a counter function using closure.

Note: A closure **"closes over"** its surrounding variables and keeps them alive. This is a important topic as many frameworks such as React, Vue and others use this concept. If you need to know more about closures please look at the tutorial of BroCode: https://youtu.be/beZfCfiuIkA?si=SnwYLGO2M7fHl6OP

```
function createCounter() {
  let count = 0;
  return function() { /* Function is returned to the variable as s reference and copies the entire environment as the count variable is from its outer scope. Note, there are similarities by appearance to a class and private variables, but they are not the same. */
    return ++count;
  };
}
const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

---

52. Immediately Invoked Function Expression (IIFE)

Task: Create a IIFE function expression

JS:

```
(function () {
  console.log("IIFE executed");
})();
```

---

53. Recursion – Factorial

Task: Create a function that calculates the **factorial** of a number n using **recursion**.

```
function factorial(n) {
  if (n === 0) return 1;
  return n * factorial(n - 1);
}
console.log(factorial(5)); // 120
```

---

54. Debouncing Function

Task: Write a debounce function that will rate limit the amount the function is called

```
function debounce(fn, delay) {
  let timer;
  return function (...args) {
    clearTimeout(timer);
    timer = setTimeout(() => fn.apply(this, args), delay);
  };
}
// Implementation to limit the search on a searchbox
const debouncedSearch = debounce(function(query) {
 console.log("Searching for:", query);
}, 300);

// Simulating typing
debouncedSearch("a");
debouncedSearch("ab");
debouncedSearch("abc"); // Only this triggers the actual search, 300ms after last call
```

---

55. Throttle Function

```
function throttle(fn, delay) {
```

```
  let last = 0;
  return function (...args) {
    let now = Date.now();
    if (now - last >= delay) {
      fn.apply(this, args);
      last = now;
    }
  };
}
```

---

56. Promisify a Callback Function

```
function asyncAdd(a, b, callback) {
  setTimeout(() => callback(null, a + b), 1000);
}
function promisifiedAdd(a, b) {
  return new Promise((resolve, reject) => {
    asyncAdd(a, b, (err, res) => {
      if (err) reject(err);
      else resolve(res);
    });
  });
}
promisifiedAdd(2, 3).then(console.log); // 5
```

---

57. Async/Await with Delay

```
function delay(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
```

```
}

async function run() {
  console.log("Start");
  await delay(1000);
  console.log("After 1 second");
}
run();
```

---

58. Custom Iterator

```
let range = {
  from: 1,
  to: 3,
  [Symbol.iterator]() {
    let current = this.from;
    let to = this.to;
    return {
      next() {
        return current <= to
          ? { done: false, value: current++ }
          : { done: true };
      }
    };
  }
};
```

```
for (let num of range) {
  console.log(num); // 1, 2, 3
}
```

---

59. Generator Function

```
function* generatorFunc() {
  yield 1;
  yield 2;
  yield 3;
}
for (let val of generatorFunc()) {
  console.log(val);
}
```

---

60. Using this in Objects and Arrow Functions

```
const obj = {
  value: 100,
  regular: function () {
    return this.value;
  },
  arrow: () => {
    return this.value;
  }
};
console.log(obj.regular()); // 100
console.log(obj.arrow());   // undefined
```

---

61. Object.freeze()

```
const user = Object.freeze({ name: "Alex" });
user.name = "John"; // Will not change
console.log(user.name); // Alex
```

---

62. Deep Clone an Object

```
let original = { a: 1, b: { c: 2 } };
let clone = JSON.parse(JSON.stringify(original));
clone.b.c = 99;
console.log(original.b.c); // 2
```

---

63. Create a Custom Promise

```
const myPromise = new Promise((resolve, reject) => {
  let success = true;
```

```
  if (success) resolve("Success!");
  else reject("Failure!");
});

myPromise.then(console.log).catch(console.error);
```

---

64. Chain Promises

```
Promise.resolve(2)
  .then(n => n * 2)
  .then(n => n + 1)
  .then(console.log); // 5
```

---

65. Dynamic Import (ES Modules)

```
// Assume utils.js exports a function
// import { helper } from './utils.js';

async function loadUtils() {
  const module = await import('./utils.js');
  module.helper();
}
```

---

66. Class and Inheritance

```
class Animal {
  speak() {
    console.log("Animal speaks");
  }
}
```

```
class Dog extends Animal {
  speak() {
    console.log("Dog barks");
  }
}
new Dog().speak(); // Dog barks
```

---

67. Static Methods in Classes

```
class MathUtil {
  static square(n) {
    return n * n;
  }
}
console.log(MathUtil.square(4)); // 16
```

---

68. Private Class Fields

```
class Counter {
  #count = 0;
  increment() {
    this.#count++;
    return this.#count;
  }
}
let c = new Counter();
console.log(c.increment()); // 1
```

---

69. Module Pattern

```
const Counter = (() => {
  let count = 0;
  return {
    increment: () => ++count,
    reset: () => count = 0
  };
})();
console.log(Counter.increment()); // 1
```

---

70. Memoization Function

```
function memoize(fn) {
  const cache = {};
  return function(n) {
    if (n in cache) return cache[n];
    return cache[n] = fn(n);
  };
}
const fib = memoize(n => n <= 1 ? n : fib(n - 1) + fib(n - 2));
console.log(fib(10)); // 55
```

---

71. Optional Chaining

```
const user = { profile: { name: "Ana" } };

console.log(user?.profile?.name); // Ana

console.log(user?.address?.city); // undefined
```

---

72. Nullish Coalescing Operator

```
let username = null;

console.log(username ?? "Guest"); // Guest
```

---

73. Symbol and Unique Keys

```
let sym = Symbol("id");

let obj = { [sym]: 123 };

console.log(obj[sym]); // 123
```

---

74. WeakMap Usage

```
let obj = {};

let wm = new WeakMap();

wm.set(obj, "secret");

console.log(wm.get(obj)); // secret
```

---

75. Event Delegation

```
document.getElementById("list").addEventListener("click", function(e) {
  if (e.target.tagName === "LI") {
    console.log("Clicked:", e.target.textContent);
  }
});
```

These are small, practical JavaScript projects designed to consolidate everything you've learned — from DOM manipulation to asynchronous logic.

JavaScript Mini Projects – 76 to 90 (with Full Code Samples)

Each mini project includes:

- ✅ Goal
- ✅ Core features
- ✅ Complete HTML + JavaScript code
- ✅ Enhancements you can try (Where applicable)

---

76. To-Do List App

Goal: Add, delete, and mark tasks as complete.

Core features: The application can do CRUD operations on the tasks as a simple TODO application

Complete HTML + JavaScript code:

```
<input id="task" type="text" placeholder="New task" />

<button onclick="addTask()">Add</button>

<ul id="todoList"></ul>


<script>

function addTask() {

  let taskInput = document.getElementById("task");

  let taskText = taskInput.value.trim();

  if (!taskText) return;
```

```
  let li = document.createElement("li");
  li.textContent = taskText;

  li.addEventListener("click", () => li.classList.toggle("done"));

  let removeBtn = document.createElement("button");
  removeBtn.textContent = "✕";
  removeBtn.onclick = () => li.remove();

  li.appendChild(removeBtn);
  document.getElementById("todoList").appendChild(li);
  taskInput.value = "";
}
</script>

<style>
 li.done { text-decoration: line-through; }
 li button { margin-left: 10px; }
</style>
```

---

77. Simple Calculator

Goal: Add or subtract from two numbers as a simple calculator

Core features: The core features of the calculator is to add and subtract two numbers from each other

Complete HTML + JavaScript code:

```
<input id="n1" type="number">
<input id="n2" type="number">
<button onclick="calc('+')">+</button>
<button onclick="calc('-')">-</button>
<p id="result"></p>

<script>
function calc(op) {
  let a = parseFloat(document.getElementById("n1").value);
  let b = parseFloat(document.getElementById("n2").value);
  let result;
  if (op === "+") result = a + b;
  if (op === "-") result = a - b;
  document.getElementById("result").textContent = "Result: " + result;
}
</script>
```

---

78. Character Counter

```
<textarea id="textArea" oninput="countChars()" rows="4"></textarea>
<p>Characters: <span id="charCount">0</span></p>

<script>
function countChars() {
  let text = document.getElementById("textArea").value;
  document.getElementById("charCount").textContent = text.length;
```

```
}
</script>
```

---

79. Live Clock

Goal: Display a live clock that updates each second

Core features: Display a live clock, updates each second

Complete HTML + JavaScript code:

```
<p id="clock"></p>


<script>
function updateClock() {
  let now = new Date().toLocaleTimeString();
  document.getElementById("clock").textContent = now;
}
setInterval(updateClock, 1000);
updateClock();
</script>
```

---

80. Tip Calculator

Goal: Calculate a tip amount for a given bill total provided

Core features: Interface to enter a bill amount and calculate the tip amount for total

Complete HTML + JavaScript code:

```
<input type="number" id="bill" placeholder="Bill Amount" />
<input type="number" id="tip" placeholder="Tip %" />
<button onclick="calculateTip()">Calculate</button>
<p id="totalTip"></p>
```

```
<script>
function calculateTip() {
  let bill = parseFloat(document.getElementById("bill").value);
  let tip = parseFloat(document.getElementById("tip").value);
  let total = bill * (tip / 100);
  document.getElementById("totalTip").textContent = "Tip: $" + total.toFixed(2);
}
</script>
```

---

81. Simple Image Slider

Goal: Display images in a simple image slider

Core features: Simple next previous buttons image slider

Complete HTML + JavaScript code:

```
<img id="slider" src="https://via.placeholder.com/200" width="200">
<button onclick="prev()">←</button>
<button onclick="next()">→</button>

<script>
let images = [
  "https://via.placeholder.com/200",
  "https://via.placeholder.com/200/ff0000",
  "https://via.placeholder.com/200/00ff00"
];
let current = 0;
```

```
function show() {
  document.getElementById("slider").src = images[current];
}

function prev() {
  current = (current - 1 + images.length) % images.length;
  show();
}

function next() {
  current = (current + 1) % images.length;
  show();
}
</script>
```

Enhancements you can try

Try to add a timer, so cycle the array of images every 1 and a half seconds.

---

82. Simple Chat Interface (Static)

Goal: Build a simple chat interface that can print the message that you typed

Core features: The interface has the ability to send a simple chat message to what you typed

Complete HTML + JavaScript code:

```
<input id="msgInput" placeholder="Type message" />
<button onclick="send()">Send</button>
<div id="chatBox"></div>

<script>
```

```
function send() {
  let msg = document.getElementById("msgInput").value;
  if (!msg.trim()) return;

  let div = document.createElement("div");
  div.textContent = msg;
  document.getElementById("chatBox").appendChild(div);
  document.getElementById("msgInput").value = "";
}
</script>
```

Enhancements you can try

Try to create a chat bot that can be hosted that two or more participants can participate in the chat. Go as far as to build a proper message queue as an advanced exercise.

---

83. Dark/Light Mode Toggle

Goal: Button to toggle light and dark mode in interface

Core features: Ability to toggle light and dark mode

Complete HTML + JavaScript code:

```
<button onclick="toggleTheme()">Toggle Theme</button>

<script>
function toggleTheme() {
  document.body.classList.toggle("dark");
}
</script>

<style>
```

```
body.dark { background: black; color: white; }
</style>
```

---

84. Accordion FAQ

Goal: Generate an accordion FAQ panel

Core features: Panel with clickable FAQ items

Complete HTML + JavaScript code:

```
<div onclick="toggleAnswer()">Q: What is JS?</div>
<div id="answer" style="display:none;">A: A scripting language</div>

<script>
function toggleAnswer() {
  let ans = document.getElementById("answer");
  ans.style.display = ans.style.display === "none" ? "block" : "none";
}
</script>
```

---

85. Form Validation (Required Fields)

```
<form onsubmit="return validate()">
  <input id="email" placeholder="Email" />
  <button type="submit">Submit</button>
</form>

<script>
function validate() {
```

```
  let email = document.getElementById("email").value;
  if (!email.includes("@")) {
    alert("Invalid email");
    return false;
  }
  return true;
}
</script>
```

---

86. Word Counter

Goal: Textarea to enter text that will calculate the amount of words entered

Core features: Counts words of text provided and show this in a text field

Complete HTML + JavaScript code:

```
<textarea id="inputText" oninput="countWords()"></textarea>
<p>Words: <span id="wordCount">0</span></p>

<script>
function countWords() {
  let text = document.getElementById("inputText").value.trim();
  let count = text ? text.split(/\s+/).length : 0;
  document.getElementById("wordCount").textContent = count;
}
</script>
```

---

87. Simple Weather UI (Static)

Goal: Show weather conditions for city provided (can be hard coded)

Core features: Counts words of text provided and show this in a text field

Complete HTML + JavaScript code:

```
<input id="city" placeholder="City Name" />

<button onclick="showWeather()">Get Weather</button>

<p id="weatherOutput"></p>


<script>

function showWeather() {

  let city = document.getElementById("city").value;

  document.getElementById("weatherOutput").textContent =

    `Weather in ${city}: Sunny 25°C`;

}

</script>
```

Enhancements you can try

Try to take these enhancements by making the city selectable and querying a weather service. If you want to make it even more attractive show weather condition icons.

---

88. Random Quote Generator

Goal: Show weather conditions for city provided (can be hard coded)

Core features: Counts words of text provided and show this in a text field

Complete HTML + JavaScript code:

```
<button onclick="newQuote()">New Quote</button>

<p id="quote"></p>


<script>
```

```
let quotes = ["Stay hungry.", "Be humble.", "Never give up."];

function newQuote() {
  let random = Math.floor(Math.random() * quotes.length);
  document.getElementById("quote").textContent = quotes[random];
}
</script>
```

---

89. Countdown Timer

Goal: A timer that counts down from a numeric value given

Core features: Counts down from a numeric value given to zero

Complete HTML + JavaScript code:

```
<input id="seconds" type="number" />
<button onclick="startTimer()">Start</button>
<p id="countdown"></p>

<script>
function startTimer() {
  let time = parseInt(document.getElementById("seconds").value);
  let display = document.getElementById("countdown");

  let interval = setInterval(() => {
    display.textContent = time;
    if (time <= 0) clearInterval(interval);
    time--;
  }, 1000);
}
```

```
</script>
```

---

90. Palindrome Checker

Goal: A Palindrome Checker is a program or function that determines whether a given input (usually a word, phrase, or number) is a palindrome — meaning it reads the same forward and backward.

Core features: Checks Palindrome for the text added

Complete HTML + JavaScript code:

```
<input id="palinText" />

<button onclick="checkPalindrome()">Check</button>

<p id="palinResult"></p>


<script>

function checkPalindrome() {

  let txt = document.getElementById("palinText").value.toLowerCase().replace(/\W/g, '');

  let isPalin = txt === txt.split('').reverse().join('');

  document.getElementById("palinResult").textContent = isPalin ? "Yes!" : "No.";

}

</script>
```

JavaScript Advanced Project Section – 91 to 101

#	**Project**	**Key Concepts**
1	Event Calendar	Dates, CRUD, localStorage
2	Chat App	WebSocket, async, DOM
3	Shopping Cart	State mgmt, storage, modular JS
4	Notes App	CRUD, localStorage, markdown
5	Crypto Tracker	API, async, charts
6	User Auth Demo	Forms, JWT, secure storage
7	Calculator	Expressions, state, UI memory
8	File Upload App	FileReader, events, validation
9	Survey + Dashboard	Dynamic forms, analytics, chart rendering
10	Translator UI	APIs, i18n UI logic
11	Custom SPA	Routing, history API, components

91. Event Calendar App (Full Code)

Goal:

Create a calendar where users can:

- View the current month
- Add events by clicking a day
- View events per day
- Delete events

📁 Directory Structure (Simple)

event-calendar/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Event Calendar App</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Event Calendar</h1>
```

```
  <div id="calendar"></div>
  <div id="eventModal" class="modal">
    <div class="modal-content">
      <span id="closeModal">&times;</span>
      <h2>Add Event</h2>
      <p id="eventDate"></p>
      <input type="text" id="eventTitle" placeholder="Event Title" />
      <button id="saveEvent">Save Event</button>
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  padding: 20px;
  background: #f2f2f2;
}
#calendar {
  display: grid;
  grid-template-columns: repeat(7, 1fr);
  gap: 5px;
}
```

```
.day {
  background: #fff;
  padding: 10px;
  border: 1px solid #ddd;
  min-height: 80px;
  position: relative;
  cursor: pointer;
}
.day:hover {
  background: #e0f7fa;
}
.day span.date {
  font-weight: bold;
}
.event {
  margin-top: 5px;
  font-size: 0.85em;
  background: #2196f3;
  color: #fff;
  padding: 2px 5px;
  border-radius: 3px;
}
/* Modal */
.modal {
  display: none;
  position: fixed;
```

```
  top: 0; left: 0;
  width: 100%; height: 100%;
  background: rgba(0,0,0,0.4);
}
.modal-content {
  background: white;
  padding: 20px;
  margin: 10% auto;
  width: 300px;
  position: relative;
  border-radius: 5px;
}
#closeModal {
  position: absolute;
  top: 10px; right: 15px;
  font-size: 1.5em;
  cursor: pointer;
}
```

---

script.js

```
const calendar = document.getElementById("calendar");
const eventModal = document.getElementById("eventModal");
const closeModal = document.getElementById("closeModal");
const eventDate = document.getElementById("eventDate");
const eventTitle = document.getElementById("eventTitle");
const saveEvent = document.getElementById("saveEvent");
```

```
let selectedDate;
let events = JSON.parse(localStorage.getItem("calendarEvents")) || {};

function renderCalendar() {
  const now = new Date();
  const year = now.getFullYear();
  const month = now.getMonth();
  const daysInMonth = new Date(year, month + 1, 0).getDate();

  calendar.innerHTML = "";

  for (let day = 1; day <= daysInMonth; day++) {
    const dateStr = `${year}-${month + 1}-${day}`;
    const dayEl = document.createElement("div");
    dayEl.className = "day";
    dayEl.dataset.date = dateStr;

    dayEl.innerHTML = `<span class="date">${day}</span>`;

    if (events[dateStr]) {
      events[dateStr].forEach(event => {
        const ev = document.createElement("div");
        ev.className = "event";
        ev.textContent = event;
        dayEl.appendChild(ev);
```

```
    });
   }
   dayEl.addEventListener("click", () => openModal(dateStr));
   calendar.appendChild(dayEl);
  }
}

function openModal(dateStr) {
  selectedDate = dateStr;
  eventDate.textContent = `Date: ${dateStr}`;
  eventTitle.value = "";
  eventModal.style.display = "block";
}

closeModal.onclick = () => {
  eventModal.style.display = "none";
};

saveEvent.onclick = () => {
  const title = eventTitle.value.trim();
  if (!title) return;

  if (!events[selectedDate]) events[selectedDate] = [];
  events[selectedDate].push(title);
  localStorage.setItem("calendarEvents", JSON.stringify(events));
  eventModal.style.display = "none";
```

```
  renderCalendar();
};

window.onclick = function(e) {
  if (e.target == eventModal) eventModal.style.display = "none";
};

renderCalendar();
```

✅ **Concepts Practiced**

- DOM rendering and event handling
- Working with Date objects
- LocalStorage for persistent state
- Dynamic element creation
- Modal interactions

---

92. Real-Time Chat App (Frontend Only)

This app lets users:

- Send and receive messages
- Simulate real-time replies using setTimeout()
  (or use a WebSocket echo server)

Goals:

- Simple chat window
- User input box + send button
- Auto-scroll to new messages
- (Optional) Simulated bot responses

📁 **Directory Structure**

chat-app/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Chat App</title>
 <link rel="stylesheet" href="style.css">
</head>
<body>
```

```
  <div class="chat-container">
    <div class="chat-window" id="chatWindow"></div>
    <div class="chat-input">
      <input type="text" id="messageInput" placeholder="Type a message..." />
      <button onclick="sendMessage()">Send</button>
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: Arial, sans-serif;
  background: #f5f5f5;
  display: flex;
  justify-content: center;
  padding: 50px;
}

.chat-container {
  width: 400px;
  border: 1px solid #ccc;
  background: white;
  display: flex;
```

```
  flex-direction: column;
  border-radius: 5px;
  overflow: hidden;
}

.chat-window {
  flex: 1;
  padding: 10px;
  overflow-y: auto;
  max-height: 400px;
}

.chat-input {
  display: flex;
  border-top: 1px solid #ddd;
}

.chat-input input {
  flex: 1;
  padding: 10px;
  border: none;
  outline: none;
}

.chat-input button {
  padding: 10px 20px;
```

```
  border: none;
  background: #2196f3;
  color: white;
  cursor: pointer;
}

.message {
  margin: 5px 0;
  padding: 8px 12px;
  border-radius: 10px;
  max-width: 75%;
}

.message.user {
  background: #e0f7fa;
  align-self: flex-end;
}

.message.bot {
  background: #eeeeee;
  align-self: flex-start;
}
```

---

script.js

```
const chatWindow = document.getElementById("chatWindow");
const messageInput = document.getElementById("messageInput");
```

```
function addMessage(text, sender) {
  const msg = document.createElement("div");
  msg.classList.add("message", sender);
  msg.textContent = text;
  chatWindow.appendChild(msg);
  chatWindow.scrollTop = chatWindow.scrollHeight;
}

function sendMessage() {
  const text = messageInput.value.trim();
  if (!text) return;

  addMessage(text, "user");
  messageInput.value = "";

  // Simulated bot response
  setTimeout(() => {
    addMessage("Echo: " + text, "bot");
  }, 800);
}
```

What You Learn Here

- DOM manipulation and scrolling
- Event-driven UI
- setTimeout() for async behavior

- Input handling and sanitation
- Component styling with CSS

Bonus: Want to Use WebSocket?

Replace the setTimeout() block with:

```
const socket = new WebSocket("wss://echo.websocket.events");


socket.onmessage = (event) => {
  addMessage("Echo: " + event.data, "bot");
};


function sendMessage() {
  const text = messageInput.value.trim();
  if (!text) return;
  addMessage(text, "user");
  messageInput.value = "";
  socket.send(text);
}
```

---

93. Shopping Cart App (with Checkout)

Goals:

- Product listing page
- Add/remove items to cart
- View cart summary with total price
- Persist cart in localStorage
- Checkout simulation (clears cart)

📁 Directory Structure

shopping-cart/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Shopping Cart</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>🛒 My Store</h1>
  <div class="product-list" id="productList"></div>
```

```
  <h2>🧺 Your Cart</h2>
  <div id="cartItems"></div>
  <p>Total: $<span id="total">0.00</span></p>
  <button onclick="checkout()">Checkout</button>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  padding: 20px;
  background: #f7f7f7;
}

.product-list {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(160px, 1fr));
  gap: 20px;
}

.product {
  background: white;
  border: 1px solid #ddd;
  padding: 10px;
```

```
  text-align: center;
  border-radius: 5px;
}

.product img {
  width: 100px;
  height: 100px;
  object-fit: contain;
}

button {
  margin-top: 10px;
  padding: 6px 12px;
  background: #2196f3;
  color: white;
  border: none;
  cursor: pointer;
  border-radius: 4px;
}

#cartItems {
  margin: 10px 0;
  padding: 10px;
  background: white;
  border: 1px solid #ddd;
}
```

---

script.js

```
const products = [
  { id: 1, name: "Laptop", price: 899, image: "https://via.placeholder.com/100" },
  { id: 2, name: "Headphones", price: 199, image: "https://via.placeholder.com/100" },
  { id: 3, name: "Keyboard", price: 99, image: "https://via.placeholder.com/100" },
];

let cart = JSON.parse(localStorage.getItem("cart")) || {};

function renderProducts() {
  const productList = document.getElementById("productList");
  products.forEach(product => {
    const div = document.createElement("div");
    div.className = "product";
    div.innerHTML = `
      <img src="${product.image}" />
      <h3>${product.name}</h3>
      <p>$${product.price.toFixed(2)}</p>
      <button onclick="addToCart(${product.id})">Add to Cart</button>
    `;
    productList.appendChild(div);
  });
}

function addToCart(productId) {
```

```
  cart[productId] = (cart[productId] || 0) + 1;
  updateCart();
}

function removeFromCart(productId) {
  if (cart[productId]) {
    cart[productId]--;
    if (cart[productId] === 0) {
      delete cart[productId];
    }
    updateCart();
  }
}

function updateCart() {
  localStorage.setItem("cart", JSON.stringify(cart));
  const cartItems = document.getElementById("cartItems");
  const totalEl = document.getElementById("total");
  cartItems.innerHTML = "";

  let total = 0;

  for (const id in cart) {
    const product = products.find(p => p.id == id);
    const quantity = cart[id];
    const subtotal = product.price * quantity;
```

```
    total += subtotal;

    const item = document.createElement("div");
    item.innerHTML = `
      ${product.name} x ${quantity} - $${subtotal.toFixed(2)}
      <button onclick="removeFromCart(${id})">❌</button>
    `;
    cartItems.appendChild(item);
  }

  totalEl.textContent = total.toFixed(2);
}

function checkout() {
  if (Object.keys(cart).length === 0) {
    alert("Your cart is empty.");
    return;
  }

  alert("Checkout complete! Thanks for your order.");
  cart = {};
  updateCart();
}

renderProducts();
updateCart();
```

Concepts You Practice

- Object manipulation
- DOM generation
- Persistent storage (localStorage)
- Stateful UI
- Function composition
- User interaction flows

---

94. In-Browser Notes App

Features

- Add, edit, and delete notes
- Persist notes in localStorage
- Optional tags or colors
- Responsive layout

📁 **Directory Structure**

notes-app/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <title>Notes App</title>

 <link rel="stylesheet" href="style.css" />

</head>

<body>

 <h1>📝 My Notes</h1>

 <div class="note-input">

  <textarea id="noteText" placeholder="Write your note..."></textarea>
```

```
    <button onclick="saveNote()">Save Note</button>
  </div>

  <div id="notesContainer" class="notes-container"></div>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  background: #f4f4f4;
  padding: 20px;
}

h1 {
  text-align: center;
}

.note-input {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-bottom: 20px;
}
```

```
.note-input textarea {
  width: 100%;
  max-width: 500px;
  height: 100px;
  padding: 10px;
  font-size: 1em;
  border-radius: 5px;
}

.note-input button {
  margin-top: 10px;
  padding: 10px 20px;
  background: #2196f3;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.notes-container {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(220px, 1fr));
  gap: 15px;
}
```

```
.note {
  background: white;
  padding: 15px;
  border-radius: 5px;
  position: relative;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.note button {
  position: absolute;
  top: 8px;
  right: 8px;
  background: #f44336;
  border: none;
  color: white;
  padding: 3px 6px;
  font-size: 0.8em;
  border-radius: 3px;
  cursor: pointer;
}
```

---

script.js

```
let notes = JSON.parse(localStorage.getItem("notes")) || [];

function saveNote() {
  const textarea = document.getElementById("noteText");
```

```
  const text = textarea.value.trim();
  if (!text) return;

  const note = {
    id: Date.now(),
    content: text
  };
  notes.push(note);
  localStorage.setItem("notes", JSON.stringify(notes));
  textarea.value = "";
  renderNotes();
}

function deleteNote(id) {
  notes = notes.filter(note => note.id !== id);
  localStorage.setItem("notes", JSON.stringify(notes));
  renderNotes();
}

function renderNotes() {
  const container = document.getElementById("notesContainer");
  container.innerHTML = "";

  notes.forEach(note => {
    const div = document.createElement("div");
    div.className = "note";
```

```
    div.innerHTML = `
      <button onclick="deleteNote(${note.id})">✕</button>
      <p>${note.content}</p>
    `;
    container.appendChild(div);
  });
}
renderNotes();
```

Concepts Covered

- Create/Read/Delete (CRUD) functionality
- Working with arrays and objects
- Using localStorage for persistence
- DOM manipulation
- Responsive layout with CSS Grid

---

95. Crypto Price Tracker with Chart.js

Goals:

- Track the current price of selected cryptocurrencies
- Display a chart showing price history
- Refresh live data using API
- Use Chart.js for visualizations

📁 Directory Structure

```
crypto-tracker/
├── index.html
├── style.css
└── script.js
```

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Crypto Price Tracker</title>
  <link rel="stylesheet" href="style.css" />
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
  <h1>📈 Crypto Price Tracker</h1>
```

```
  <select id="coinSelect">
    <option value="bitcoin">Bitcoin</option>
    <option value="ethereum">Ethereum</option>
    <option value="dogecoin">Dogecoin</option>
  </select>
  <button onclick="fetchCryptoData()">Refresh</button>

  <h2 id="priceDisplay">Price: Loading...</h2>

  <canvas id="priceChart" width="600" height="300"></canvas>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  background: #f5f5f5;
  padding: 20px;
  text-align: center;
}

select, button {
  padding: 10px;
  margin: 10px;
```

```
  font-size: 16px;
}

canvas {
  margin-top: 20px;
  background: white;
  border-radius: 5px;
  box-shadow: 0 2px 6px rgba(0,0,0,0.1);
}
```

---

script.js

```
let chart;
const coinSelect = document.getElementById("coinSelect");
const priceDisplay = document.getElementById("priceDisplay");

async function fetchCryptoData() {
  const coin = coinSelect.value;
  const apiURL = `https://api.coingecko.com/api/v3/coins/${coin}/market_chart?vs_currency=usd&days=1`;

  try {
    const res = await fetch(apiURL);
    const data = await res.json();

    const prices = data.prices.map(p => ({
      time: new Date(p[0]).toLocaleTimeString(),
```

```
      price: p[1]
    }));

    const latestPrice = prices[prices.length - 1].price;
    priceDisplay.textContent = `Price: $${latestPrice.toFixed(2)}`;

    renderChart(prices);
  } catch (err) {
    priceDisplay.textContent = "Failed to fetch data.";
    console.error(err);
  }
}

function renderChart(prices) {
  const labels = prices.map(p => p.time);
  const data = prices.map(p => p.price);

  if (chart) chart.destroy();

  const ctx = document.getElementById("priceChart").getContext("2d");
  chart = new Chart(ctx, {
    type: "line",
    data: {
      labels: labels,
      datasets: [{
        label: "Price (USD)",
```

```
      data: data,
      borderColor: "#2196f3",
      backgroundColor: "rgba(33, 150, 243, 0.1)",
      tension: 0.2,
      fill: true
    }]
  },
  options: {
    responsive: true,
    scales: {
      x: { ticks: { maxTicksLimit: 8 } },
      y: { beginAtZero: false }
    }
  }
 });
}

// Load default data on startup
fetchCryptoData();
```

Concepts Practiced

- Fetching JSON from REST APIs
- Parsing and processing external data
- Chart.js integration for visual feedback
- Time and price formatting
- Responsive, data-driven UI

API Used

- CoinGecko API (Free, No Auth)
    - Example Endpoint:
    - https://api.coingecko.com/api/v3/coins/bitcoin/market_chart?vs_currency=usd&days=1

---

96. User Auth Demo (Mock JWT)

Goals:

- Simulated login with hardcoded credentials
- Store and use a fake JWT token in localStorage
- Logout functionality
- Protected view that only shows when logged in

📁 Directory Structure

auth-demo/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>User Auth Demo</title>
 <link rel="stylesheet" href="style.css" />
</head>
<body>
 <div class="container">
  <h1>User Auth Demo</h1>
```

```
  <div id="authSection">
    <input type="text" id="username" placeholder="Username" />
    <input type="password" id="password" placeholder="Password" />
    <button onclick="login()">Login</button>
  </div>

  <div id="protectedSection" style="display: none;">
    <p>Welcome, <span id="userDisplay"></span>! You are logged in.</p>
    <button onclick="logout()">Logout</button>
  </div>
 </div>

 <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
 font-family: sans-serif;
 background: #f0f0f0;
 padding: 40px;
}

.container {
 max-width: 400px;
 margin: 0 auto;
```

```
  background: white;
  padding: 20px;
  border-radius: 8px;
  text-align: center;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
}

input {
  display: block;
  width: 100%;
  padding: 10px;
  margin: 10px 0;
  font-size: 1em;
}

button {
  padding: 10px 20px;
  background: #2196f3;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
```

---

script.js

```
const authSection = document.getElementById("authSection");
```

```
const protectedSection = document.getElementById("protectedSection");
const userDisplay = document.getElementById("userDisplay");

// Hardcoded credentials
const fakeUser = {
  username: "admin",
  password: "1234"
};

// Simulate JWT creation
function generateToken(username) {
  return btoa(`${username}:${Date.now()}`); // base64 encode
}

// Check for saved token
function checkAuth() {
  const token = localStorage.getItem("token");
  const username = localStorage.getItem("username");

  if (token && username) {
    authSection.style.display = "none";
    protectedSection.style.display = "block";
    userDisplay.textContent = username;
  } else {
    authSection.style.display = "block";
    protectedSection.style.display = "none";
```

```
  }
}

// Login logic
function login() {
  const inputUser = document.getElementById("username").value.trim();
  const inputPass = document.getElementById("password").value;

  if (inputUser === fakeUser.username && inputPass === fakeUser.password) {
    const token = generateToken(inputUser);
    localStorage.setItem("token", token);
    localStorage.setItem("username", inputUser);
    checkAuth();
  } else {
    alert("Invalid credentials!");
  }
}

// Logout logic
function logout() {
  localStorage.removeItem("token");
  localStorage.removeItem("username");
  checkAuth();
}

// Initial check on load
```

checkAuth();

✅ **Concepts Practiced**

- Authentication flow
- Working with localStorage
- DOM manipulation
- Mock JWT and token management
- Conditional UI rendering

Note:

This is **not secure** for real-world use — it's a learning exercise. For real apps, you'd:

- Validate credentials on a server
- Use real JWTs (with exp, iat, sub, etc.)
- Protect routes via tokens

---

97. Calculator with History (No eval())

Goals:

- Basic arithmetic operations (+ − × ÷)
- Clickable UI buttons for digits/operators
- Real-time display update
- Calculation history stored in localStorage
- Option to clear history

📁 **Directory Structure**

calculator-app/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <title>Calculator with History</title>
 <link rel="stylesheet" href="style.css" />
</head>
<body>
 <div class="calculator">
  <div id="display">0</div>
```

```
<div class="buttons">
  <button onclick="press('7')">7</button>
  <button onclick="press('8')">8</button>
  <button onclick="press('9')">9</button>
  <button onclick="press('/')">÷</button>

  <button onclick="press('4')">4</button>
  <button onclick="press('5')">5</button>
  <button onclick="press('6')">6</button>
  <button onclick="press('*')">×</button>

  <button onclick="press('1')">1</button>
  <button onclick="press('2')">2</button>
  <button onclick="press('3')">3</button>
  <button onclick="press('-')">−</button>

  <button onclick="press('0')">0</button>
  <button onclick="press('.')">.</button>
  <button onclick="calculate()">=</button>
  <button onclick="press('+')">+</button>

  <button onclick="clearDisplay()">C</button>
</div>

<h3>📜 History</h3>
<ul id="history"></ul>
```

```
    <button onclick="clearHistory()">Clear History</button>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  background: #f4f4f4;
  display: flex;
  justify-content: center;
  padding: 40px;
}

.calculator {
  background: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  width: 300px;
}

#display {
  background: #222;
```

```
  color: lime;
  font-size: 2em;
  padding: 10px;
  text-align: right;
  border-radius: 4px;
  margin-bottom: 10px;
  overflow-x: auto;
}

.buttons {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 10px;
}

button {
  padding: 15px;
  font-size: 1em;
  border: none;
  background: #2196f3;
  color: white;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
```

```
  background: #1976d2;
}

ul#history {
  padding-left: 20px;
  font-size: 0.9em;
  max-height: 150px;
  overflow-y: auto;
}
```

---

script.js

```
let input = "";
const display = document.getElementById("display");
const historyList = document.getElementById("history");

function press(value) {
  input += value;
  display.textContent = input;
}

function clearDisplay() {
  input = "";
  display.textContent = "0";
}

function calculate() {
```

```
  try {
    // Simple parser for basic arithmetic
    const result = Function('"use strict";return (' + input + ')')();
    if (isNaN(result)) throw "Invalid Expression";
    const record = `${input} = ${result}`;
    addToHistory(record);
    display.textContent = result;
    input = result.toString();
  } catch {
    display.textContent = "Error";
    input = "";
  }
}

function addToHistory(entry) {
  let history = JSON.parse(localStorage.getItem("calcHistory")) || [];
  history.unshift(entry);
  if (history.length > 10) history.pop();
  localStorage.setItem("calcHistory", JSON.stringify(history));
  renderHistory();
}

function renderHistory() {
  const history = JSON.parse(localStorage.getItem("calcHistory")) || [];
  historyList.innerHTML = "";
  history.forEach(item => {
```

```
    const li = document.createElement("li");
    li.textContent = item;
    historyList.appendChild(li);
  });
}

function clearHistory() {
  localStorage.removeItem("calcHistory");
  renderHistory();
}

renderHistory();
```

Concepts Practiced

- Expression parsing with Function() (safer than eval)
- LocalStorage for persistent state
- DOM manipulation and state tracking
- UI input sanitization and error handling
- History and array management

---

98. File Upload & Preview App

Goals:

- Upload multiple image files
- Preview thumbnails immediately
- Validate file type and size
- Option to remove a preview

📁 **Directory Structure**

file-upload-preview/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Image Upload Preview</title>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <div class="container">
    <h1>📷 Upload & Preview Images</h1>
    <input type="file" id="fileInput" multiple accept="image/*" />
```

```
    <div id="previewContainer" class="preview-container"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  background: #f5f5f5;
  padding: 20px;
  text-align: center;
}

.container {
  background: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  max-width: 700px;
  margin: auto;
}

input[type="file"] {
  margin: 20px 0;
}
```

```
.preview-container {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  justify-content: center;
}

.preview {
  position: relative;
  border: 1px solid #ccc;
  padding: 5px;
  border-radius: 5px;
}

.preview img {
  height: 100px;
  width: auto;
  border-radius: 4px;
}

.preview button {
  position: absolute;
  top: 2px;
  right: 2px;
  background: #f44336;
```

```
  border: none;
  color: white;
  font-size: 0.8em;
  border-radius: 50%;
  cursor: pointer;
  width: 20px;
  height: 20px;
}
```

---

script.js

```
const fileInput = document.getElementById("fileInput");
const previewContainer = document.getElementById("previewContainer");

fileInput.addEventListener("change", handleFiles);

function handleFiles(event) {
  const files = event.target.files;
  previewContainer.innerHTML = ""; // Clear previous previews

  for (const file of files) {
    if (!file.type.startsWith("image/")) {
      alert(`${file.name} is not an image.`);
      continue;
    }

    const reader = new FileReader();
```

```
    reader.onload = function(e) {
      const div = document.createElement("div");
      div.className = "preview";

      const img = document.createElement("img");
      img.src = e.target.result;

      const btn = document.createElement("button");
      btn.textContent = "×";
      btn.onclick = () => div.remove();

      div.appendChild(img);
      div.appendChild(btn);
      previewContainer.appendChild(div);
    };

    reader.readAsDataURL(file);
  }
}
```

Concepts Practiced

- File input handling
- FileReader API (readAsDataURL)
- Dynamic DOM creation
- Image previewing with memory-safe URLs
- Input validation and error feedback

99. Survey Builder + Analytics Dashboard

Goals:

- Create survey questions (text or multiple choice)
- Collect responses via form
- Analyze answers and show simple charts
- Save questions and responses in localStorage

📁 Directory Structure

survey-dashboard/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Survey Builder & Dashboard</title>
 <link rel="stylesheet" href="style.css" />
</head>
<body>
 <div class="container">
  <h1>📋 Survey Builder</h1>
  <input id="questionInput" placeholder="Enter your question" />
```

```
    <button onclick="addQuestion()">Add Question</button>
    <ul id="questionList"></ul>

    <h2>👱 Respond to Survey</h2>
    <form id="surveyForm"></form>
    <button onclick="submitSurvey()">Submit Response</button>

    <h2>📊 Dashboard</h2>
    <div id="dashboard"></div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  background: #f9f9f9;
  padding: 20px;
}

.container {
  max-width: 700px;
  margin: auto;
  background: white;
```

```
  padding: 20px;
  border-radius: 8px;
}

input, button {
  padding: 10px;
  margin: 5px 0;
  width: 100%;
  font-size: 1em;
}

ul {
  list-style: none;
  padding: 0;
}

li {
  background: #eee;
  margin: 5px 0;
  padding: 10px;
  border-radius: 4px;
}

form div {
  margin-bottom: 10px;
}
```

```
#dashboard div {
  margin: 10px 0;
  padding: 10px;
  background: #e3f2fd;
  border-radius: 4px;
}
```

---

script.js

```
let questions = JSON.parse(localStorage.getItem("surveyQuestions")) || [];
let responses = JSON.parse(localStorage.getItem("surveyResponses")) || [];

const questionInput = document.getElementById("questionInput");
const questionList = document.getElementById("questionList");
const surveyForm = document.getElementById("surveyForm");
const dashboard = document.getElementById("dashboard");

function saveData() {
  localStorage.setItem("surveyQuestions", JSON.stringify(questions));
  localStorage.setItem("surveyResponses", JSON.stringify(responses));
}

function renderQuestions() {
  questionList.innerHTML = "";
  surveyForm.innerHTML = "";
```

```
  questions.forEach((q, index) => {
    const li = document.createElement("li");
    li.textContent = q;
    questionList.appendChild(li);

    // Add input to survey form
    const div = document.createElement("div");
    div.innerHTML = `<label>${q}</label><br><input type="text" name="q${index}" required />`;
    surveyForm.appendChild(div);
  });
}

function addQuestion() {
  const text = questionInput.value.trim();
  if (!text) return;
  questions.push(text);
  questionInput.value = "";
  saveData();
  renderQuestions();
}

function submitSurvey() {
  const formData = new FormData(surveyForm);
  const answerObj = {};
  questions.forEach((q, i) => {
```

```
    answerObj[q] = formData.get(`q${i}`);
  });
  responses.push(answerObj);
  saveData();
  renderDashboard();
  surveyForm.reset();
  alert("Thanks for your response!");
}

function renderDashboard() {
  dashboard.innerHTML = "";
  if (responses.length === 0) {
    dashboard.textContent = "No responses yet.";
    return;
  }

  questions.forEach(question => {
    const answerList = responses.map(r => r[question]);
    const div = document.createElement("div");
    div.innerHTML = `<strong>${question}</strong><br>${answerList.join(", ")}`;
    dashboard.appendChild(div);
  });
}

renderQuestions();
renderDashboard();
```

Concepts Covered

- Dynamic form generation from data
- Multi-question survey handling
- Storing structured data in localStorage
- Building an analytics dashboard (summary view)
- DOM rendering and form processing

---

100. Multi-Language Translator UI

Goals:

- Translate user-entered text from one language to another
- Fetch results from an external translation API
- Support dropdowns for selecting source and target languages
- Display translated result instantly
- Error handling for bad input/API issues

📁 Directory Structure

translator-app/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <title>Language Translator</title>
 <link rel="stylesheet" href="style.css" />
</head>
<body>
 <div class="container">
  <h1>🌍 Language Translator</h1>
```

```
<textarea id="inputText" placeholder="Enter text to translate..."></textarea>

<div class="controls">
 <select id="fromLang">
  <option value="en">English</option>
  <option value="es">Spanish</option>
  <option value="fr">French</option>
  <option value="de">German</option>
  <option value="hi">Hindi</option>
 </select>

 <span>→</span>

 <select id="toLang">
  <option value="es">Spanish</option>
  <option value="en">English</option>
  <option value="fr">French</option>
  <option value="de">German</option>
  <option value="hi">Hindi</option>
 </select>
</div>

<button onclick="translateText()">Translate</button>

<div id="resultBox">
```

```
    <h3>Translation:</h3>
    <p id="resultText">–</p>
   </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: sans-serif;
  background: #f1f1f1;
  padding: 20px;
  text-align: center;
}

.container {
  background: white;
  padding: 20px;
  margin: auto;
  max-width: 600px;
  border-radius: 10px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}
```

```
textarea {
  width: 100%;
  height: 100px;
  margin-bottom: 15px;
  padding: 10px;
  font-size: 1em;
}

.controls {
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 10px;
  margin-bottom: 15px;
}

select {
  padding: 8px;
  font-size: 1em;
}

button {
  padding: 10px 20px;
  background: #2196f3;
  color: white;
  border: none;
```

```
  font-size: 1em;
  border-radius: 5px;
  cursor: pointer;
}

#resultBox {
  margin-top: 20px;
  text-align: left;
}

#resultText {
  background: #e3f2fd;
  padding: 10px;
  border-radius: 5px;
  white-space: pre-line;
}
```

---

script.js

```
const inputText = document.getElementById("inputText");
const resultText = document.getElementById("resultText");
const fromLang = document.getElementById("fromLang");
const toLang = document.getElementById("toLang");

async function translateText() {
  const text = inputText.value.trim();
  const from = fromLang.value;
```

```
const to = toLang.value;

if (!text) {
  alert("Please enter text to translate.");
  return;
}

resultText.textContent = "Translating...";

try {
  const res = await fetch("https://libretranslate.de/translate", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      q: text,
      source: from,
      target: to,
      format: "text"
    })
  });

  const data = await res.json();
  resultText.textContent = data.translatedText || "Translation failed.";
} catch (err) {
  resultText.textContent = "Error: Could not fetch translation.";
  console.error(err);
```

```
 }
}
```

Concepts Covered

- fetch() API with POST
- JSON formatting and headers
- Async/await and error handling
- DOM updates from API response
- Language selection dropdown logic

Translation API Used

- **LibreTranslate (Free, Open Source)**
    - Endpoint: https://libretranslate.de/translate
    - Docs: https://libretranslate.com

⚠️ This is public and **rate-limited**, so it's good for testing or learning. For production, you'd host your own LibreTranslate instance or use a paid service.

---

101. Single Page App (Custom Router)

Features

- Navigation between “pages” without reloading
- History management using pushState and popstate
- A simple route-to-view mapping
- Clean, modular code

📁 **Directory Structure**

spa-router/

├── index.html

├── style.css

└── script.js

---

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <title>Simple SPA Router</title>
 <link rel="stylesheet" href="style.css" />
</head>
<body>
 <nav>
  <a href="/" data-link>Home</a>
  <a href="/about" data-link>About</a>
```

```
  <a href="/contact" data-link>Contact</a>
 </nav>

 <div id="app">
  <!-- Content will be injected here -->
 </div>

 <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
 font-family: sans-serif;
 margin: 0;
 padding: 0;
}

nav {
 background: #2196f3;
 padding: 10px;
 text-align: center;
}

nav a {
 color: white;
```

```
  text-decoration: none;
  margin: 0 15px;
  font-weight: bold;
}

#app {
  padding: 20px;
}
```

---

script.js

```
// Define your routes
const routes = {
  "/": () => `
    <h1>🏠 Home</h1>
    <p>Welcome to our simple SPA!</p>
  `,
  "/about": () => `
    <h1><b>i</b> About</h1>
    <p>This is a custom client-side routed page.</p>
  `,
  "/contact": () => `
    <h1>📞 Contact</h1>
    <p>Reach out to us at contact@example.com</p>
  `
};
```

```
// Handle navigation
function navigateTo(url) {
  history.pushState(null, null, url);
  router();
}
// Render route
function router() {
  const path = location.pathname;
  const route = routes[path] || (() => `<h1>404</h1><p>Page not found</p>`);
  document.getElementById("app").innerHTML = route();
}
// Set up event listeners
window.addEventListener("popstate", router);
document.addEventListener("DOMContentLoaded", () => {
  document.body.addEventListener("click", e => {
    if (e.target.matches("[data-link]")) {
      e.preventDefault();
      navigateTo(e.target.href);
    }
  });
  router();
});
```

Concepts Covered

- **Client-side routing** with pushState and popstate
- Route-to-view mapping using JS objects

- Dynamic content injection with innerHTML
- Navigation interception using event delegation
- Modular route logic without frameworks

Bonus Ideas

- Load routes from external files (e.g., using fetch)
- Use hash-based routing (#/home) for simpler setups
- Add page transitions or animations

Great question! Let's break down and implement those **bonus improvements** to the custom SPA router project (Project 11). These upgrades will make your routing more flexible, interactive, and beginner-friendly.

🔧 **Bonus Improvements for SPA Router Code**

✅ **1. Load Routes from External Files via fetch()**

Let's assume we store HTML snippets in individual files in a pages/ folder:

spa-router/

├── pages/

│ ├── home.html

│ ├── about.html

│ └── contact.html

Update routes to:

```
const routes = {
  "#/": "pages/home.html",
  "#/about": "pages/about.html",
  "#/contact": "pages/contact.html"
```

```
};
```

Updated router() function:

```
async function router() {
  const path = location.hash || "#/";
  const page = routes[path] || "pages/404.html";

  try {
    const res = await fetch(page);
    const html = await res.text();
    document.getElementById("app").innerHTML = html;
  } catch (err) {
    document.getElementById("app").innerHTML = "<h1>Error loading page</h1>";
  }
}
```

---

2. Use Hash-Based Routing (for Simpler Deployment)

Hash-based routing is easier to work with because it doesn't require server-side rewrites for clean URLs.

HTML changes:

Update links like this:

```
<nav>
  <a href="#/">Home</a>
  <a href="#/about">About</a>
  <a href="#/contact">Contact</a>
</nav>
```

Key Benefit:

You don’t need to handle server-side 404s or route fallback logic — everything after # is handled by JavaScript only.

---

3. Add Page Transitions or Animations

Let’s use simple **fade-in animations** when switching views.

Add this to style.css:

```
#app {
  opacity: 0;
  transition: opacity 0.3s ease-in-out;
}

#app.fade-in {
  opacity: 1;
}
```

Modify router() to apply the class:

```
async function router() {
  const path = location.hash || "#/";
  const page = routes[path] || "pages/404.html";

  try {
    const res = await fetch(page);
    const html = await res.text();
    const app = document.getElementById("app");

    app.classList.remove("fade-in");
```

```
    app.innerHTML = html;

    // Force reflow to restart animation
    void app.offsetWidth;
    app.classList.add("fade-in");

  } catch {
    document.getElementById("app").innerHTML = "<h1>Error loading page</h1>";
  }
}
```

Full Integration Tips

Feature	Benefit
fetch() external files	Keep logic + content separated
Hash routing #/page	Works on all static servers
fade-in animations	Improves user experience
