Chapter 8

Forms: are elements that control such as input fields, select menus and buttons.

* <form>

A Searching Example: an input field with a search query

Accessing form elements: The legacy DOM had a useful property document.forms that returns HTML collection.

* An index notation to return the first (and only) form object
  + Const form = document.forms[0];

Input Fields: are the most common types of form control

Text input fields: is used for entering a short piece of text, such as username.

Password input fields: enter passwords or secret information.

Checkbox input fields: select different options that can be checked for true, or left unchecked as false.

Radio button input fields: provide exclusive chose of options, only one option can be selected.

Hidden input fields: These are not displayed by the browser, but have a 'value' attribute that can contain information that is submitted with the form. They are often used to send information such as settings or information that the user has already provided.

Form Validation: is the process of checking whether a suer has entered the information into a form correctly.

* A required field is completed
* An email address is valid
* A number is entered when numerical data is required
* A password is at least a minimum number of characters

Validation can occur on the client side using JavaScript, and on the server side. It is advisable to use both client-side and server-side validation.

Disabling the Submit Button: Another useful technique that can aid usability is to disable the submit button if there are errors on the form

If an input field is empty, the following function will disable

* function disableSubmit(event) {
* if(event.target.value === ''){
* document.getElementById('submit').disabled = true;
* } else {
* document.getElementById('submit').disabled = false;
* }
* }

Chapter 12:

Object-oriented programming is often used to model representations of objects in the real world.

Encapsulation: the inner workings are kept hidden inside the object and only the essential functionalities are exposed to the end user, such as the 'on' button.

Polymorphism: this means various objects can share the same method, but also have the ability to override shared methods with a more specific implementation

Inheritance: In OOP, this means we can take an object that already exists and inherit all its properties and methods

Classes: Objects are then created as an instance of that class, and inherit all the properties and methods of the class.

Static Methods: The static keyword can be used in class declarations to create a static method. These are sometimes called class methods in other programming languages. A static method is called by the class directly rather than by instances of the class.

Polymorphism: that different objects can have the same method, but implement it in different ways.

* [1,2,3].toString()
* << '1,2,3'
* 2..toString; // remember 2 dot operators for integers!
* << '2'

Mixins: is a way of adding properties and methods of some Objects to another object without using inheritance.

Chatper 15:

Modular JavaScript: module is a self-contained piece of code that provides functions and methods that can then be used in other files and by other modules. This helps to keep code organized in separate, reusable files, which improves code maintainability. The code in a module should have a single purpose, and group together functions with distinct functionality.

Sanbox:

Chapter 6, A Vector Type

* class Vec {
* constructor(x, y) {
* this.x = x;
* this.y = y;
* }
* plus(other) {
* return new Vec(this.x + other.x, this.y + other.y);
* }
* minus(other) {
* return new Vec(this.x - other.x, this.y - other.y);
* }
* get length() {
* return Math.sqrt(this.x \* this.x + this.y \* this.y);
* }
* }
* console.log(new Vec(1, 2).plus(new Vec(2, 3)));
* // → Vec{x: 3, y: 5}
* console.log(new Vec(1, 2).minus(new Vec(2, 3)));
* // → Vec{x: -1, y: -1}
* console.log(new Vec(3, 4).length);
* // → 5
* {x: 3, y: 5}
* {x: -1, y: -1}
* 5

Chapter 10: Roads module:

const {buildGraph} = require("./graph");

* const roads = [
* "Alice's House-Bob's House", "Alice's House-Cabin",
* "Alice's House-Post Office", "Bob's House-Town Hall",
* "Daria's House-Ernie's House", "Daria's House-Town Hall",
* "Ernie's House-Grete's House", "Grete's House-Farm",
* "Grete's House-Shop", "Marketplace-Farm",
* "Marketplace-Post Office", "Marketplace-Shop",
* "Marketplace-Town Hall", "Shop-Town Hall"
* ];
* exports.roadGraph = buildGraph(roads.map(r => r.split("-")));