DAY 1: INTRODUCTION TO HTML,CSS

DATE: 11-06-2024

Contents Covered:

Day 1: Understanding HTML Basics

- **What I Learned**:

HTML stands for Hyper Text Markup Language.

It is the backbone of any webpage, providing structure and content.

Key components include:

Elements: The building blocks of HTML (e.g., <p>, <h1>,).

Attributes: Additional information about elements (e.g., class, id).

Common tags include:

- <html>: The root element.
- <head>: Contains metadata like title and links.
- <body>: The visible content of the webpage.
- <h1> to <h6>: Headings.
- <p>: Paragraphs.
- <a>: Hyperlinks.

Practice:

Created a basic HTML file with a title, headings, and a paragraph.

Day 2: Digging Deeper into HTML

DATE: 12-06-2024

What I Learned:

HTML lists:

Ordered lists (<ol>) for numbered items.

Unordered lists (<ul>) for bulleted items.

List items (<li>) for each list entry.

Adding images:

<img> tag with src for the image path and alt for description.

Forms and inputs:

<form> for creating forms.

<input>, <textarea>, <button>, and <select> for user inputs.

Practice:

Built a simple webpage with a list of favorite books, an image, and a feedback form.

Day 3: Introduction to CSS

DATE: 13-06-2024

What I Learned:

CSS stands for Cascading Style Sheets.

It styles HTML elements, including layout, colors, and fonts.

Ways to use CSS:

Inline: Inside the style attribute in an HTML tag.

Internal: Inside a <style> tag in the <head>.

External: In a separate .css file linked with <link>.

Basic syntax:

Selectors (e.g., h1, .class, #id) define the elements to style.

Declaration blocks contain properties (e.g., color, font-size) and values.

Practice:

Styled headings and paragraphs with colors and font sizes.

Experimented with inline, internal, and external CSS.

Day 4: CSS Styling and Layout Basics

DATE: 14-07-2024

What I Learned:

Styling basics:

color, background-color, font-family, and font-size for text.

Box model concepts: margin, border, padding, and width.

Layouts:

display property: block, inline, flex, etc.

position property: static, relative, absolute, fixed.

CSS Grid and Flexbox for responsive layouts.

Practice:

Created a webpage with a styled header, navigation bar, and content area.

Used Flexbox to align items horizontally.

Day 5: Putting It All Together

DATE: 17-07-2024

What I Learned:

Combining HTML and CSS to build a complete webpage.

Importance of semantic HTML (e.g., <header>, <footer>, <article>) for better structure and accessibility.

Debugging CSS using browser developer tools.

Practice:

Designed a personal portfolio webpage with:

A header and navigation menu.

A section for projects.

A footer with contact details.

Day 6: Introduction to JavaScript Basics

DATE: 18-06-2024

What I Learned Today

Today was my first step into JavaScript, a language that brings life and interactivity to web pages. I started with understanding the basics of the language:

What is JavaScript?

JavaScript is a high-level, interpreted programming language.

It runs in web browsers and allows developers to create dynamic content like image sliders, form validations, and interactive menus.

Variables and Their Types

Variables are containers that store data. JavaScript allows three ways to declare variables:

var: The older way to declare variables, function-scoped.

let: Modern, block-scoped, used for variables that may change.

const: Block-scoped, used for variables that won't change.

JavaScript Data Types

Primitive Data Types: These include strings, numbers, booleans, undefined, null, and symbol.

Non-Primitive Data Types: Objects and arrays.

Basic Operators

Arithmetic Operators: +, -, *, /, % for basic math operations.

Comparison Operators: ==, ===, <, >, etc., to compare values.

Logical Operators: && (AND), || (OR), ! (NOT) for combining conditions.

New Insights

JavaScript is case-sensitive, so let Name and let name are treated as different variables.

Using === instead of == for comparison is safer as it checks both value and type.

Day 7: Functions and Control Flow

DATE: 19-06-2024

What I Learned Today

Functions are the building blocks of reusable code. Today, I focused on:

1. Functions in JavaScript

- A function is a reusable block of code that performs a specific task. Functions help to avoid redundancy and improve cod code maintainability.

- Functions can be declared using the `function` keyword.

- Functions can also return values.

2. Control Flow and Conditional Statements

- I studied how to make decisions in JavaScript using `if`, `else if`, and `else` statements. These allow you to execute different blocks of code based on conditions.

- The `switch` statement is another way to handle multiple conditions more efficiently.

3. Logical Operators

- I explored logical operators: **`&&` (AND)**, **`||` (OR)**, and **`!` (NOT)**. These are used to combine or negate conditions.

Day 8: Arrays and Loops

DATE: 20-06-2024

What I Learned Today

Arrays are essential for storing multiple values in a single variable. Today, I learned:

1. Arrays in JavaScript

- Arrays store multiple values of the same type (or mixed types) in an ordered manner.

2. Array Methods

- `push()` and `pop()` add or remove elements from the end of an array.
- `shift()` and `unshift()` add or remove elements from the beginning of an array.
- `concat()` joins two or more arrays into one.
- `slice()` extracts a portion of an array.

3. Loops

- I explored different ways to iterate over arrays:
  - For loop: Useful for iterating a fixed number of times.
  - `for...of` loop: Better for iterating over the values of an array.
  - `forEach()` method: A functional approach to iterate over an array.

Day 9: Objects and Their Properties

DATE: 21-06-2024

What I Learned Today

Objects are crucial for organizing related data. I focused on:

1. Objects in JavaScript

- An object is a collection of key-value pairs, where keys are strings, and values can be any data type.

2. Accessing and Modifying Object Properties

- Properties can be accessed using dot notation or bracket notation.

3. Nested Objects

- Objects can contain other objects as properties.

Practical Exercises and Problem-Solving

1. Task 1: Store a Book's Information

I created a book object and printed its details.

2. Task 2: Add a Method to an Object

I added a method to the book object to toggle its availability.

Day 10: JavaScript StringS

DATE: 24-06-2024

What I Learned Today

Strings are one of the most commonly used data types. I explored:

1. String Methods

- **`charAt()`:** Returns the character at a given index.
- **`toUpperCase()` and `toLowerCase()`:** Convert the case of a string.
- **`slice()`:** Extracts a section of a string.
- **`replace()`:** Replaces parts of the string.

2. Template Literals

- I used template literals to embed variables inside strings using backticks**.**

Practical Exercises and Problem-Solving

1. Task 1: Reverse a String

I used `split()`, `reverse()`, and `join()` to reverse a string.

2. Task 2: Count Vowels in a String

I wrote a function to count the vowels in a string using `match()` and regular expressions.

- String methods are powerful tools for manipulating text.

- Template literals are an excellent way to build strings with variables embedded, making the code cleaner and easier to read.

Day 11: Introduction to Object-Oriented Programming (OOP)**

DATE: 25-06-2024

What I Learned Today

Today, I took a deep dive into Object-Oriented Programming (OOP), a key paradigm used in JavaScript. OOP is about structuring code in a way that models real-world entities as objects. I covered the following concepts:

1. Classes and Objects

- A class is a blueprint for creating objects with common properties and methods.

- An object is an instance of a class.

2. Encapsulation

- Encapsulation is the concept of bundling data (properties) and methods (functions) that operate on the data into a single unit called a class.

- I also learned how to make some properties or methods private by using `#` (in newer versions of JavaScript).

3. Inheritance

- Inheritance allows a class to inherit properties and methods from another class.

- The extends keyword is used for inheritance.

Practical Exercises and Problem-Solving

1. Task 1: Create a Book Class

I created a class to model a book with properties like `title`, `author`, and `pages` and methods to display the book's details.

2. Task 2: Create a BankAccount Class with Inheritance I created a `BankAccount` class and a `SavingsAccount` subclass that inherits from it.

Day 12: Error Handling and Debugging

DATE: 27-06-2024

What I Learned Today

Today was focused on handling errors and debugging JavaScript code. I explored how to anticipate and handle issues gracefully to make code more robust.

1. Types of Errors

Syntax Errors: Mistakes in the syntax of the code (e.g., missing parentheses or semicolons).

Runtime Errors: Errors that occur while the code is running (e.g., calling a function on `undefined`).

Logical Errors: Errors where the code runs but doesn't behave as expected.

2. Error Handling with Try-Catch

- The `try` block is used to execute code that might throw an error, and the `catch` block catches the error if one occurs.

- The `finally` block always runs, regardless of whether an error occurred or not.

3. Throwing Custom Errors

- We can throw custom errors using the `throw` statement.

4. Debugging with Console Methods

- I used `console.log()` for simple debugging.

Practical Exercises and Problem-Solving

1. Task 1: Divide by Zero Error Handling

I created a function that divides two numbers and uses `try-catch` to handle divide-by-zero errors**.**

Day 13: Asynchronous JavaScript (Callbacks)

DATE: 28-06-2024

What I Learned Today

I dove into the asynchronous nature of JavaScript. Since many tasks (e.g., network requests) can take time, JavaScript needs to handle these operations without freezing the application.

1. Callbacks

- A callback is a function that is passed as an argument to another function and is executed when the task is complete.

- This allows JavaScript to perform non-blocking operations. While waiting for one task to complete, the program can execute other code.

2. Asynchronous Nature with `setTimeout()`

- `setTimeout()` is a built-in function that runs a piece of code after a specified delay (in milliseconds). This simulates asynchronous behavior.

Practical Exercises and Problem-Solving

1. Task 1: Delayed Message

I wrote a function that prints a message after a delay using `setTimeout()`.

2. Task 2: Using Callbacks in Data Fetching Simulation

I simulated fetching data asynchronously using a callback.

Day 14: Promises and Async/Await

DATE: 01-07-2024

What I Learned Today

I continued exploring asynchronous programming with Promises and Async/Await, two powerful techniques for handling asynchronous code.

1. Promises

- A Promise is an object representing the eventual completion (or failure) of an asynchronous operation.

- Promises can be in one of three states: pending, fulfilled, or rejected.

2. Async/Await

- `async` is used to declare a function as asynchronous, and `await` pauses the execution of the function until the Promise resolves.

Practical Exercises and Problem-Solving

1. Task 1: Simulate a Delay with Promises

I created a function that simulates a delay using a Promise.

2. Task 2: Use Async/Await with Promises

I used async/await to handle the promise result.

Day 15: JavaScript DOM Manipulation (Introduction)

DATE: 02-07-2024

What I Learned Today

Today was all about learning how to interact with the Document Object Model (DOM) in JavaScript. The DOM is a programming interface for web documents, representing the page structure as a tree of objects that can be manipulated. Here are the key topics I covered:

1. What is the DOM?

- The DOM represents an HTML document as a tree of nodes, where each element, attribute, and text is a node.

- JavaScript can be used to traverse, modify, and manipulate this tree dynamically.

2. Accessing DOM Elements

- `document.getElementById()`: Selects an element by its `id`.

- `document.getElementsByClassName()`: Selects elements by class name.

- `document.querySelector()`: Selects the first element that matches a CSS selector.

- `document.querySelectorAll()`: Selects all elements that match a CSS selector.

3. **Creating and Adding Elements**

- Creating a new element.

- Appending the element to the DOM:

Practical Exercises and Problem-Solving

1. Task 1: Update Heading Text

2. Task 2: Add a New List Item

Day 16: Introduction to React

DATE: 03-07-2024

What I Learned Today

Today, I started learning about React, a JavaScript library for building user interfaces. I understood that React is primarily used to create Single Page Applications (SPAs) and relies on components for reusability and modularity. A component is essentially a small, self-contained piece of code that renders part of the UI. React uses JSX (JavaScript XML), which is a syntax extension allowing us to write HTML directly within JavaScript.

React applications are built using a hierarchy of components. The root component (usually App.js) is the starting point, and other components are nested within it. React embraces declarative programming, focusing on what the UI should look like rather than how to achieve it procedurally.

React's approach to building UI using components is both fascinating and efficient. The introduction to JSX made me appreciate how it combines HTML and JavaScript seamlessly, making UI design more intuitive.

Tasks for Today

Install Node.js and set up a React project using npx create-react-app my-app.

Run the React development server with npm start.

Modify the default App.js file to display "Hello, React!".

Reflect on the structure of a React app and the role of each file (e.g., index.js, App.js).

Day 17: React Components

DATE: 04-07-2024

What I Learned Today

React applications are built from components, which can be either functional or class-based. Functional components are simpler and defined as JavaScript functions that return JSX. Class components, on the other hand, use ES6 classes and can manage their own state.

Props (short for properties) are how we pass data from one component to another. Props are immutable, making the flow of data unidirectional in React. This approach enhances predictability and maintainability.

Learning about components and props reinforced the idea of React's modular structure. The ability to pass data dynamically via props is a powerful tool for creating reusable components.

Tasks for Today

Create a functional component that accepts props to display a personalized greeting.

Write a class component that logs a message in the console when rendered.

Understand the difference between functional and class components through experimentation.

Day 18: State Management in React

What I Learned Today

DATE: 05-07-2024

DAY: Thursday

State is a built-in object used to track data that changes over time. In functional components, state is managed using the useState hook. State updates cause React to re-render the component, ensuring the UI reflects the latest data. React state is local to the component where it is defined but can be passed as props to child components.

State management is crucial for building interactive applications. The useState hook simplifies state handling and eliminates the need for class components in many cases.

Tasks for Today

Create a counter app that increments and decrements the count using useState.

Add a reset button to reset the count to zero.

Experiment with managing multiple state variables in a single component.

Day 19: React Lifecycle Methods and useEffect

DATE: 08-07-2024

What I Learned Today

Lifecycle methods are a feature of class components, but in functional components, they are replaced by the useEffect hook. useEffect handles side effects like data fetching, subscriptions, and manual DOM manipulations. It runs after the component renders and can be controlled by specifying dependencies.

The useEffect hook is incredibly versatile and simplifies handling side effects in functional components. Understanding dependencies is key to optimizing performance and avoiding unwanted re-renders.

Tasks for Today

Create a timer that counts seconds using useEffect.

Add a start/stop button to control the timer.

Implement cleanup logic to stop the timer when the component unmounts.

Day 20: useReducer Hook in React

DATE: 09-07-2024

What I Learned Today

The `useReducer` hook is an alternative to `useState` for managing more complex state logic. It is especially useful when dealing with state that involves multiple sub-values or requires sophisticated state transitions. `useReducer` works by dispatching actions to a reducer function, which determines how the state should be updated based on the action type. It provides better structure for managing complex state logic in React applications.

useReducer is helpful when state updates become complex or involve multiple actions. It organizes the logic in a clearer manner, making it easier to maintain as the application grows.

Tasks for Today

1. Implement a counter app using `useReducer`.

2. Modify the reducer function to handle multiple types of actions (reset, multiply, etc.).

3. Experiment with managing form state using `useReducer` instead of `useState`.

Day 21: useContext Hook in React

DATE: 10-07-2024

DAY: Tuesday

What I Learned Today

The `useContext` hook is used to access the value of a context directly within a component. It provides an easier and more direct way of consuming context values in functional components, without the need for a `Consumer` wrapper. This is especially useful when passing data through many layers of components (avoiding props drilling).

useContext simplifies how we access shared data across components. It is a great alternative to props drilling and makes passing data easier when working with deeply nested components.

Tasks for Today

1. Create a theme toggle using the `useContext` hook.

2. Refactor a component that needs to access a global value (e.g., user authentication status) using `useContext`.

3. Experiment with context nesting and dynamic updates.

Day 22: useRef Hook in React

DATE: 11-07-2024

What I Learned Today

The `useRef` hook allows us to create mutable references to DOM elements or other values that persist across renders. This is helpful for interacting with the DOM directly, like accessing input fields, or keeping track of previous state or props without triggering a re-render. It returns a mutable `ref` object that can hold any value.

`useRef` is a powerful hook for handling DOM references and mutable values that need to persist across renders. It's an excellent tool for interacting with the DOM directly in React without affecting the component state.

Tasks for Today

1. Create an input field and use `useRef` to focus it programmatically.

2. Use `useRef` to store a value that persists across renders without triggering a re-render.

3. Experiment with accessing and manipulating DOM elements using `useRef`.