MAD 1 WEEK 11

Beyond HTML - Detailed Lecture Notes

HTML Evolution

Markup Languages

- **Origins**: Emerged in the late 1960s for typesetting and document management.
- **Primary Use**: Creating structured documents for print and other forms of media.
- **Challenges**:
    - Lack of standardization across implementations.
    - Varied target audience: coders, publishers, and academics.
    - Output formats not universally adaptable (e.g., print focus).
    - Limited machine readability and adaptability.

SGML (Standard Generalized Markup Language)

- Introduced as a meta-language to define other markup languages.

Key Features

1. **Declarative**:
    - Focus on defining structure and attributes without specifying how to process the document.
2. **Rigorous**:
    - Ensures strict structure, comparable to database design.

Document Type Definition (DTD)

- A formal specification for families of markup languages within SGML.
- Allows customization of tags and document behavior based on needs.

Applications

- Early markup languages and their derivatives used SGML for structure and extensibility.

HTML (HyperText Markup Language)

Origins and Evolution

- Initially an application of SGML but designed to be forgiving of errors for accessibility.
- **HTML Timeline**:
    - **HTML 2.0**: Early attempts to align with SGML compliance.
    - **HTML 4.0**: Became officially SGML-compliant but saw limited usage.
    - **HTML5**:
        - Broke away from SGML.
        - Defined its own parsing rules for better flexibility and modern needs.

Key Characteristics

- **Parsing**: Designed to be lenient to facilitate broader adoption.
- **Backward Compatibility**: Retains support for older tags and features while modernizing functionalities.
- **HTML5 Maintenance**: Now a **living standard** maintained by the **WHATWG (Web Hypertext Application Technology Working Group)**.

XML (eXtensible Markup Language)

Overview

- A simplified derivative of SGML designed for extensibility and versatility.
- Enables developers to define custom tags for specific use cases.

Key Features

- **Simplicity**: Easy to use and understand.
- **Human-Readable & Machine-Readable**: Designed to balance usability for both.
- **Structure**: Supports complex data relationships and structures.

Applications

- Widely used for data interchange and presentation.
- Examples include:
    - **RSS Feeds**: News and blog syndication.
    - **MathML**: Mathematical notation.
    - **SVG**: Scalable vector graphics.

XHTML (eXtensible HyperText Markup Language)

Overview

- Reformulation of HTML4 as an XML-based application.
- Aimed to clean up inconsistencies in HTML4.

Key Features

- Modular structure for better organization.
- More extensible than traditional HTML.
- **XML Namespaces**: Allow interoperability with other XML-based applications.

HTML5

Introduction

- The final version of HTML.
- Focused on adding modern features like multimedia, canvas, and app support while maintaining simplicity.

Key Differences from SGML

- No longer an SGML or XML application.
- Defines its own parser and rules, independent of SGML standards.

HTML5 Maintenance

- Managed as a **living standard** by WHATWG.
- Continuously updated to adapt to evolving web technologies.

Features

- Multimedia support (audio, video).
- Enhanced APIs for better browser interaction.
- Modern elements for semantic web development.

Extending HTML

Custom Elements

- Allow developers to define new tags and functionalities.
- Achieved using JavaScript APIs.
- Challenges:
    - May lead to inconsistent semantics if not standardized.
    - Risk of misuse due to open-ended tag creation.

Web Components

1. **Custom Elements**: Extend existing tags or create new ones.
2. **Shadow DOM**: Isolates styles and scripts for component encapsulation.

3. **HTML Templates**: Provides reusable and structured templates for content.

Benefits

- Promotes code reuse and modular design.
- Allows developers to create self-contained, customizable components.

JavaScript and Modern Web Development

What is JavaScript?

Overview

- **High-level programming language**: Offers features that simplify coding for humans.
    - **Dynamic Typing**: Variables can hold values of different types without explicit declaration.
    - **Object Orientation**: Prototype-based, rather than classical (as in Java).
- **Multi-paradigm**: Supports multiple programming styles:
    - **Event-driven**: Handles asynchronous events efficiently (e.g., user clicks).
    - **Functional**: Functions are first-class citizens; allows functional composition.
    - **Imperative**: Uses statements and procedures to directly perform computations.

Why Learn JavaScript?

- **Browser Integration**: Built into most web browsers via dedicated JavaScript engines (e.g., V8 for Chrome).
- **Robust APIs**: Provides a wide range of tools for development:
    - Text and date manipulation, regular expressions.
    - Standard data structures like dictionaries.
    - **DOM Manipulation**: Enables real-time interaction with web pages.
- **No Native IO**: Files and system-level access are handled through external APIs.

Strengths in Web Development

- **Core Use**: DOM manipulation to dynamically alter the browser environment.
- **Relatively Easy**: Syntax and structure have similarities to Python, Java, and C/C++ (though unrelated).

Learning Resources

- MDN JavaScript Basics

- Learn JavaScript Online

Custom Elements and Web Components

Custom Elements

- **Definition**: HTML5 allows developers to define their own tags and behavior via JavaScript APIs.
- **Considerations**:
  - **Meaning**: Does the custom tag represent a clear function (e.g., `<my-button>`)?
  - **Rendering**: How should the element be displayed?
  - **State**: Like built-in elements (`<input>`), custom elements may need state management.
  - Can be:
    - **Customized Built-in Elements**: Extend existing tags (e.g., `<button is="my-button">`).
    - **Autonomous Custom Elements**: Fully new, standalone tags (e.g., `<my-component>`).

Web Components

- **Key Features**:
  - **Custom Elements**: Define new, reusable HTML tags with extended functionality.
  - **Shadow DOM**: Encapsulates styling and behavior, ensuring no interference with global styles.
  - **HTML Templates**: Use `<template>` and `<slot>` for reusable and structured content.

Examples and Resources

- Web Components Examples on MDN
- Tutorials: Editable List Example, Word Count Component.

Goals

- Promote **code reuse** and **modularity**.
- **Challenge**: Limited standardization across implementations.

Frameworks in Web Development

Purpose

- **Solve Common Problems**: Reduce repetitive coding tasks and standardize approaches.
- **Examples of Boilerplate Solutions**:
    - **Flask**: Python web framework for API/backend development.
    - **React**: JavaScript library for frontend component management.

Advantages

- **Code Efficiency**: Avoid reinventing the wheel.
- **Standard Techniques**: Use widely accepted design patterns for web development.

Single Page Applications (SPAs)

Definition

- Applications that load a single HTML page and dynamically update content without reloading.
- Improves user experience with faster, more seamless interaction.

Framework Support

- JavaScript frameworks like **React**, **Angular**, and **Vue** are tailored for SPA development.

Popular Frameworks Overview

1. **React**
    - **Library**: Primarily for building user interfaces.
    - **Declarative**: Developers specify "what" the UI should look like, not "how" it works.
    - **Components**:
        - Similar to Web Components but more declarative.
        - Focuses on UI composition and state management.
    - **Popularity**: Widely used due to its simplicity and versatility.
    - Learn React.
2. **Angular**
    - Developed and maintained by Google.
    - A **comprehensive framework**: Provides everything from templating to state management.
    - Best suited for large-scale applications with complex requirements.
3. **Vue.js**
    - Lightweight and beginner-friendly.
    - Combines the best features of React and Angular.

    - Great for small to medium-sized projects.
4. **Ember.js**
    - Focuses on convention-over-configuration.
    - Combines component-based design with service integration.
    - Suitable for highly structured and scalable applications.

Comparison Resource

- Client-Side JavaScript Frameworks Introduction (MDN)

Summary

- **JavaScript**: Core language for dynamic, interactive web development.
- **Custom Elements & Web Components**: Extend HTML5 capabilities for modular, reusable designs.
- **Frameworks**: Streamline development for SPAs, reducing boilerplate and improving efficiency.
- **React, Angular, and Vue**: The most popular choices for modern web applications.

Javascript Basics:

1. Setting Up and Using JavaScript

Inline JavaScript: Add JavaScript directly into an HTML element.

```
<button onclick="alert('Hello, World!')">Click Me</button>
```

External File: Link an external `.js` file using `<script>`:

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

---

2. Variables

Variables store data. JavaScript uses `var`, `let`, or `const` to declare variables.

Declaration Types:

- **`let`**: Block-scoped, reassignable.
- **`const`**: Block-scoped, not reassignable (constant).
- **`var`**: Function-scoped, outdated (avoid using).

Example:

```
let name = "Alice"; // Can be reassigned
const pi = 3.14;    // Cannot be reassigned
var age = 25;       // Avoid using
```

---

3. Data Types

1. **Primitive**:
    - Number: `42`, `3.14`
    - String: `"Hello"`
    - Boolean: `true`, `false`
    - Null: `null` (intentional absence of value)
    - Undefined: A variable declared but not assigned.
2. **Objects**: Complex data structures like arrays and custom objects.

Example:

```
let number = 10;              // Number
let text = "JavaScript!";     // String
let isFun = true;             // Boolean
let nothing = null;           // Null
let notAssigned;              // Undefined
```

---

4. Operators

Arithmetic: `+`, `-`, `*`, `/`, `%`

Comparison: `==`, `===`, `!=`, `<`, `>`

Logical: `&&` (AND), `||` (OR), `!` (NOT)

Example:

```
let x = 10;
let y = 5;

console.log(x + y);  // 15
console.log(x > y);  // true
console.log(x === y); // false
```

---

5. Functions

Reusable blocks of code that perform specific tasks.

Example:

```
function greet(name) {
  return `Hello, ${name}!`;
}

console.log(greet("Alice")); // Output: Hello, Alice!
```

---

6. Control Structures

Conditional Statements

```
let age = 20;

if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
```

Loops

For Loop:

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

While Loop:

```
let count = 0;
while (count < 5) {
  console.log(count);
```

```
  count++;
}
```

---

7. Arrays

Collections of values.

Example:

```
let fruits = ["Apple", "Banana", "Cherry"];

console.log(fruits[1]); // Banana
fruits.push("Date");    // Add to the end
fruits.pop();           // Remove last element
```

---

8. Objects

Key-value pairs for storing related data.

Example:

```
let person = {
  name: "Alice",
  age: 25,
  greet: function () {
    console.log(`Hi, I'm ${this.name}`);
  }
};

console.log(person.name); // Alice
person.greet();           // Hi, I'm Alice
```

---

9. Events

React to user interactions like clicks, typing, etc.

Example:

```
document.getElementById("myButton").addEventListener("click",
function() {
```

```
  alert("Button clicked!");
});
```

---

10. Document Object Model (DOM)

JavaScript can manipulate HTML and CSS via the DOM.

Examples:
Select Element:

```
let heading = document.querySelector("h1");
heading.textContent = "Hello, JavaScript!";
```

- **Change Style**:
  ```
  heading.style.color = "blue";
  ```