Wdp (DI01032011) - Problem Solver

Antigravity AI

June 4, 2026

Contents

CHAPTER 1

Web Development Basics

1.1 Web Site Development Process

Methodology:

Website Development Lifecycle 1. **Information Gathering**: Identify the purpose, goals, and target audience of the website. 2. **Planning**: Create a sitemap and define the technology stack required for the project. 3. **Design**: Create wireframes and mockups to establish the visual layout and user interface. 4. **Development**: Write HTML, CSS, and JavaScript code to build the functional web pages. 5. **Testing**: Test the website on different browsers and devices for compatibility and resolve any bugs. 6. **Delivery and Maintenance**: Deploy the website to a live web server and provide ongoing updates.

Worked Example:

Develop a Lifecycle Plan for a Bakery Website **Step 1: Information Gathering**
Purpose: Sell baked goods online. Audience: Local residents and event organizers.
Step 2: Planning
Sitemap: Home | Menu | About Us | Contact.
Tech Stack: HTML and CSS.
Step 3: Design
Wireframe layout: Logo in header, horizontal navigation menu below, hero image of baked goods, main content area with featured items, and footer with contact information.
Step 4: Development
Code the Home and Menu pages using HTML tags for structure and CSS rules for styling.
Step 5: Testing
Verify that the website displays correctly on Google Chrome, Mozilla Firefox, and mobile screens. Check if all internal links are working correctly.
Step 6: Delivery
Upload the website files to a web hosting service using FTP and configure the domain `www.localbakery.com`.

1.2 Web Page Layout and Wireframing

Methodology:

Basic Wireframing Algorithm 1. **Define the Container**: Establish the main wrapper for the web page content. 2. **Add Header**: Place the header at the top (usually containing the site logo and main title). 3. **Insert Navigation**: Place navigation links either horizontally below the header or vertically in a sidebar. 4. **Define Main Content Area**: Allocate space for the primary information (articles, products, forms). 5. **Add Sidebar (Optional)**: Place secondary information or advertisements on the left or right side. 6. **Add Footer**: Place the footer at the bottom for copyright information and secondary links.

Worked Example:

Design a Wireframe Structure for a Personal Blog **Step 1: Define Container**
Set a fixed-width or responsive wrapper for the entire page to keep content centered.
Step 2: Add Header
Create a top section containing the text "My Personal Blog" and a small profile icon.
Step 3: Insert Navigation
Add a horizontal navigation bar below the header containing links: Home | Articles | About | Contact.
Step 4: Define Main Content
Allocate a left column (70% of the width) to display a list of blog posts with titles, dates, and short excerpts.
Step 5: Add Sidebar
Allocate a right column (30% of the width) containing a search box, categories list, and a brief "About Me" widget.
Step 6: Add Footer
Create a bottom section containing the copyright notice: "©2026 My Personal Blog".

1.3 Web Site Usability and Accessibility Check

Methodology:

Accessibility Compliance Procedure 1. **Check Alternative Text**: Ensure all `<img>` tags have descriptive `alt` attributes for screen readers. 2. **Verify Color Contrast**: Confirm that text color stands out sufficiently against the background color. 3. **Ensure Keyboard Navigation**: Verify that all interactive elements (links, buttons) can be accessed and triggered using the `Tab` and `Enter` keys. 4. **Semantic HTML Structure**: Use proper heading tags (`<h1>` to `<h6>`) in sequential order without skipping levels.

Worked Example:

Fix Accessibility Issues in a Web Page Snippet **Problem Statement:**
A web page contains the following elements that fail accessibility standards:

- An image of a flowchart: `<img src="flowchart.jpg">`
- Light gray text on a white background.
- A custom button created with a `<div>` that requires a mouse click.

Apply the accessibility procedure to resolve these issues.
Step 1: Check Alternative Text
Add an `alt` attribute to the image describing its content.
Fix: `<img src="flowchart.jpg" alt="Flowchart showing the login process">`
Step 2: Verify Color Contrast
Change the text color from light gray to dark gray or black to ensure high contrast against the white background.
Step 3: Ensure Keyboard Navigation
Replace the custom `<div>` button with a standard `<button>` element to natively support keyboard interaction.
Fix: Use `<button type="button">Submit</button>`.

1.4 Configuring Browser Settings

Methodology:

Procedure to Clear Browser Cache 1. **Open Settings**: Launch the web browser and open the main menu to access Settings. 2. **Navigate to Privacy**: Locate the "Privacy and Security" or "History" section. 3. **Select Clear Data**: Click on "Clear browsing data" or "Clear history". 4. **Choose Time Range**: Select the appropriate time range (e.g. "All time" or "Last hour"). 5. **Select Cache Options**: Check the box specifically for "Cached images and files". 6. **Execute**: Click the "Clear data" button and reload the webpage.

Worked Example:

Resolve Display Issues on a Development Server **Problem Statement:**
A web developer updated the CSS file for a website but the new styling changes are not reflecting when the page is reloaded. How should they configure the browser to fix this issue?
Step 1: Open Settings
Open the browser menu and go to Settings.
Step 2: Navigate to Privacy
Go to the "Privacy and Security" tab.
Step 3: Select Clear Data
Click on "Clear browsing data".
Step 4: Choose Time Range
Set the time range to "All time" to ensure all old versions are removed.
Step 5: Select Cache Options
Ensure "Cached images and files" is checked. (Uncheck passwords or browsing history if they do not need to be cleared).
Step 6: Execute
Click "Clear data". Reload the web page to force the browser to fetch the newly updated CSS file directly from the server.

1.5 Web Project Management Fundamentals

Methodology:

Work Breakdown Structure Creation 1. **Identify the Main Project**: Define the final deliverable at the top level of the hierarchy. 2. **Divide into Major Phases**: Break the project down into major lifecycle phases (e.g. Planning, Design, Implementation). 3. **Decompose Phases into Tasks**: Break down each phase into specific actionable tasks. 4. **Assign Resources and Time**: Allocate team members and estimated time to each task.

Worked Example:

Create a WBS for a Portfolio Website **Step 1: Identify the Main Project**
Project: John's Online Portfolio Website.
Step 2: Divide into Major Phases

- Phase 1: Planning
- Phase 2: Design
- Phase 3: Development

Step 3: Decompose Phases into Tasks
Phase 1: Planning

1.1 Define project goals
1.2 Register domain name and hosting
Phase 2: Design
2.1 Create page wireframes
2.2 Select color scheme and typography
Phase 3: Development
3.1 Write HTML structure
3.2 Apply CSS styling
3.3 Add JavaScript for interactive gallery
Step 4: Assign Resources and Time
Task 3.1 (HTML structure) - Assigned to Web Developer (2 Days).
Task 3.2 (CSS styling) - Assigned to UI Designer (3 Days).

1.6 Web Navigation Concepts

Methodology:

Designing a Navigation Structure 1. **Inventory Content**: List all pages and sections the website will contain. 2. **Group Content**: Categorize related pages into logical sections (e.g. placing all service pages under "Services"). 3. **Determine Navigation Type**: Choose between Top Navigation, Sidebar Navigation, or Hamburger Menu based on the screen size and content volume. 4. **Create Labels**: Assign clear and concise descriptive labels to each navigation link. 5. **Establish Hierarchy**: Define primary navigation (main menu) and secondary navigation (dropdowns or footer links).

Worked Example:

Design Navigation for an Ecommerce Store **Step 1: Inventory Content**
Pages: Home, Men's Clothing, Women's Clothing, Electronics, About Us, Contact, Privacy Policy, Terms of Service.
Step 2: Group Content

- Shop: Men's Clothing, Women's Clothing, Electronics
- Company: About Us, Contact
- Legal: Privacy Policy, Terms of Service

Step 3: Determine Navigation Type
Use a Top Horizontal Navigation bar for primary links, and a Footer Navigation for legal links.
Step 4: Create Labels
Labels: "Shop", "About", "Contact", "Legal".
Step 5: Establish Hierarchy
Primary Navigation (Header):
Home | Shop (Dropdown: Men, Women, Electronics) | About | Contact
Secondary Navigation (Footer):
Privacy Policy | Terms of Service

1.7 Web Graphics and Multimedia Optimization

Methodology:

Image Optimization Algorithm 1. **Select Appropriate Format**: Choose JPEG for photographs, PNG for images with transparency, and SVG for vector graphics (logos, icons). 2. **Resize Dimensions**: Scale the image to the maximum physical dimensions required by the webpage layout (e.g. resize a 4000px image to 800px). 3. **Compress File Size**: Run the image through a compression tool to reduce file size without significant loss of visual quality. 4. **Implement Responsive Images**: Use the `srcset` attribute in HTML to provide different image sizes for different screen resolutions.

Worked Example:

Optimize a Logo and a Hero Image for Web **Problem Statement:**
You have a 5MB company logo in JPEG format with a white background, and a 12MB photograph (6000x4000 pixels) for the homepage banner. Optimize them for the web.
Step 1: Select Appropriate Format
Logo: Convert to PNG or SVG to support a transparent background instead of white.
Photograph: Keep as JPEG but optimize it, or convert to a modern format like WebP.
Step 2: Resize Dimensions
Logo: Resize from original large dimensions to 200x50 pixels.
Photograph: Resize from 6000x4000 to 1920x1080 pixels (standard HD width for hero banners).
Step 3: Compress File Size
Run both images through a compression tool. The logo reduces to 15KB, and the photograph reduces from 12MB to 250KB.
Step 4: Implement Responsive Images
For the hero image, generate a smaller 800px version for mobile devices and use HTML `srcset` to serve the appropriate size based on the user's screen.

CHAPTER 2

Hyper Text Markup Language (HTML)

2.1 HTML Basics and Formatting Tags

Methodology:

HTML Basic Structure and Formatting To create a basic HTML document with formatted text follow these steps:

1. Start the document with the `<!DOCTYPE html>` declaration.
2. Open the `<html>` tag.
3. Define the head section using `<head>` and add metadata and a title using `<title>`.
4. Open the `<body>` tag to hold the visible content.
5. Use heading tags (`<h1>` to `<h6>`) for titles and subtitles.
6. Use the paragraph tag (`<p>`) for regular text.
7. Apply inline formatting tags such as `<b>` or `<strong>` for bold text `<i>` or `<em>` for italicized text and `<u>` for underlined text.
8. Close the `<body>` and `<html>` tags.

Worked Example:

Creating a Formatted Web Page **Problem:** Write an HTML code snippet to display a welcome message with a main heading a subheading a paragraph with some bold and italicized text and an underlined sentence. **Solution:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <h1>Welcome to Web Development</h1>
    <h2>Introduction to HTML</h2>
    <p>HTML stands for <b>Hyper Text Markup Language</b>. It is used to
    create <i>web pages</i> and web applications.</p>
    <p><u>Always remember to close your HTML tags.</u></p>
</body>
</html>
```

2.2 Lists and Hyperlinks

Methodology:

Creating Lists and Hyperlinks To structure content into lists and link to other web pages follow these steps:

1. For an unordered (bulleted) list open the `<ul>` tag.
2. For an ordered (numbered) list open the `<ol>` tag.
3. Inside the list tag use the `<li>` tag for each individual list item.
4. Close the list tag (`</ul>` or `</ol>`).
5. To create a hyperlink use the anchor tag `<a>`.
6. Set the `href` attribute in the anchor tag to the target URL for example `<a href="url">Link Text</a>`.

Worked Example:

Web Page with Lists and Navigation Links **Problem:** Create a webpage containing navigation links to "Home" and "About Us" followed by an ordered list of web technologies and an unordered list of their features.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Lists and Links</title>
</head>
<body>
    <h2>Navigation</h2>
    <p>
        <a href="home.html">Home</a> |
        <a href="about.html">About Us</a>
    </p>

    <h2>Web Technologies</h2>
    <ol>
        <li>HTML</li>
        <li>CSS</li>
        <li>JavaScript</li>
    </ol>

    <h2>Features of HTML</h2>
    <ul>
        <li>Easy to learn</li>
        <li>Platform independent</li>
        <li>Supports multimedia</li>
    </ul>
</body>
</html>
```

2.3 Designing HTML Tables

Methodology:

Building Data Tables To create a structured table to display tabular data:

1. Open the `<table>` tag. Use the `border` attribute to add visible grid lines (e.g. `<table border="1">`).
2. Define a table row using the `<tr>` tag.
3. Inside the header row use `<th>` for table header cells which render text as bold and centered by default.
4. For subsequent data rows use `<td>` for standard table data cells.
5. To merge columns horizontally use the `colspan="n"` attribute within a `<th>` or `<td>` tag.
6. To merge rows vertically use the `rowspan="n"` attribute within a `<th>` or `<td>` tag.
7. Close each row with `</tr>` and finally close the table with `</table>`.

Worked Example:

Creating a Class Time Table **Problem:** Write an HTML code to create a simple class time table for two days showing the time subject and a combined break slot that spans across two rows.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Class Time Table</title>
</head>
<body>
    <h2>Weekly Time Table</h2>
    <table border="1">
        <tr>
            <th>Day</th>
            <th>10:00 - 11:00</th>
            <th>11:00 - 12:00</th>
            <th>12:00 - 1:00</th>
        </tr>
        <tr>
            <td>Monday</td>
            <td>HTML Basics</td>
            <td>CSS Styling</td>
            <td rowspan="2">Lunch Break</td>
        </tr>
        <tr>
            <td>Tuesday</td>
            <td>JavaScript</td>
            <td>Web Forms</td>
        </tr>
    </table>
</body>
</html>
```

2.4 Creating Web Forms

Methodology:

Developing Interactive Web Forms To collect user input and submit data using HTML forms:

1. Open the `<form>` tag. Specify the `action` attribute for the destination script and the `method` attribute (GET or POST).
2. Use the `<label>` tag to describe the purpose of each input field.
3. Use the `<input>` tag for single-line text fields passwords radio buttons and checkboxes by setting the appropriate `type` attribute.
4. Use `<input type="radio">` and assign the identical `name` attribute to group mutually exclusive options.
5. Use the `<select>` and `<option>` tags to create a dropdown selection list.
6. Use the `<textarea>` tag for capturing multi-line text input.
7. Add a submission button using `<input type="submit" value="Register">`.
8. Close the form using the `</form>` tag.

Worked Example:

Creating a Student Registration Form **Problem:** Create an HTML registration form that collects a student name password gender (using radio buttons) and course selection (using a dropdown menu).
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Registration</title>
</head>
<body>
    <h2>Registration Form</h2>
    <form action="submit.html" method="POST">
        <label>Student Name:</label>
        <input type="text" name="student_name"><br><br>

        <label>Password:</label>
        <input type="password" name="student_password"><br><br>

        <label>Gender:</label>
        <input type="radio" name="gender" value="Male"> Male
        <input type="radio" name="gender" value="Female"> Female<br><br>

        <label>Select Course:</label>
        <select name="course">
            <option value="diploma">Diploma</option>
            <option value="degree">Degree</option>
        </select><br><br>

        <input type="submit" value="Register">
    </form>
</body>
```

```
</html>
```

2.5 Multimedia Inserting Techniques and Frames

Methodology:

Embedding Multimedia and I-Frames To embed images video audio files and external document frames into a web page:

1. To insert an image use the `<img>` tag with the `src` attribute pointing to the image file and the `alt` attribute for alternative textual description.
2. To embed a video use the `<video>` tag with the `controls` attribute. Place a `<source src="file.mp4" type="video/mp4">` tag inside it.
3. To embed audio use the `<audio>` tag with the `controls` attribute enclosing a `<source src="file.mp3" type="audio/mpeg">` tag.
4. To display another HTML document within the current page use the inline frame `<iframe>` tag.
5. Set the `src` attribute of the iframe to the target URL and specify the dimensions using `width` and `height` attributes.

Worked Example:

Web Page with Image and Iframe **Problem:** Write HTML code to embed a company logo image and an inline frame displaying an external website.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Multimedia and Frames</title>
</head>
<body>
    <h2>Company Logo</h2>
    <img src="logo.png" alt="Company Logo" width="200" height="100">

    <h2>External Content via Iframe</h2>
    <iframe src="https://www.example.com" width="600" height="400">
        Your browser does not support iframes.
    </iframe>
</body>
</html>
```

2.6 Semantic Tags and Metadata

Methodology:

Using Semantic Tags and Metadata To improve Search Engine Optimization (SEO) accessibility and to define a proper logical document structure:

1. Inside the `<head>` section use the `<meta>` tag to define metadata.
2. Use `<meta charset="UTF-8">` to specify character encoding.

3. Use `<meta name="description" content="...">` and `<meta name="keywords" content="...">` to assist search engines.
4. In the `<body>` section use semantic structural tags instead of generic layout tags like `<div>`.
5. Use `<header>` for the introductory content.
6. Use `<nav>` for the main navigation block.
7. Use `<main>` for the dominant content of the document.
8. Use `<section>` and `<article>` to group related thematic content together.
9. Use `<footer>` for the footer section containing copyright or contact information.

Worked Example:

Structured Web Page with Semantic Elements **Problem:** Construct an HTML page utilizing semantic tags (header nav section footer) and include basic metadata for keywords and description.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="description" content="A tutorial on HTML semantic tags">
    <meta name="keywords" content="HTML Semantic Web Development">
    <title>Semantic HTML Page</title>
</head>
<body>
    <header>
        <h1>My Personal Blog</h1>
    </header>

    <nav>
        <a href="#home">Home</a> |
        <a href="#posts">Posts</a> |
        <a href="#contact">Contact</a>
    </nav>

    <main>
        <section id="posts">
            <article>
                <h2>Understanding HTML5</h2>
                <p>HTML5 introduced many semantic elements that make
                the web page structure more meaningful.</p>
            </article>
        </section>
    </main>

    <footer>
        <p>&copy; 2024 My Personal Blog. All rights reserved.</p>
    </footer>
</body>
</html>
```

2.7 GUI HTML Editors

Methodology:

Using GUI HTML Editors Graphical User Interface (GUI) HTML editors allow developers to design web pages visually without manually coding every tag. The algorithmic workflow is as follows:

1. Open the GUI editor and create a new project or HTML file.
2. Switch to the Design or Split view interface.
3. Drag and drop web elements (such as tables forms or images) from the tool palette onto the visual canvas.
4. Use the properties panel to modify the attributes of the selected element (e.g. changing text color setting image source paths or adding borders).
5. The editor automatically generates the underlying HTML source code corresponding to the visual design.
6. Preview the page in an integrated web browser directly from the editor.
7. Switch to the Code view to manually refine or troubleshoot the auto-generated HTML snippet.

Worked Example:

Workflow for Creating a Table via GUI Editor **Problem:** Outline the visual workflow to insert a 3x3 table using a GUI HTML editor instead of writing the `<table>` tags manually.
Solution:

1. **Step 1:** Open the new HTML file in the editor and click on the visual Design workspace tab.
2. **Step 2:** Navigate to the top application menu and select `Insert` → `Table`.
3. **Step 3:** A configuration dialog box appears. Enter `3` for Rows and `3` for Columns. Set the Border thickness value to `1`.
4. **Step 4:** Click OK. The 3x3 table grid is visually rendered on the canvas.
5. **Step 5:** Click inside any table cell and type the desired data such as headers or numbers.
6. **Step 6:** Switch to the Code view to observe the editor-generated `<table>` `<tr>` and `<td>` tags representing the design.

CHAPTER 3
Cascading Style Sheets

3.1 Introduction to Cascading Style Sheets

Methodology:

Adding CSS to HTML CSS (Cascading Style Sheets) can be applied to HTML in three primary ways.

1. **Inline CSS**: Using the `style` attribute directly inside an HTML tag.
2. **Internal CSS**: Using the `<style>` tag inside the `<head>` section of the HTML document.
3. **External CSS**: Linking to an external `.css` file using the `<link>` tag inside the `<head>` section.

Worked Example:

Applying Inline CSS Write HTML code to display a heading in blue and a paragraph in red using inline CSS.
Solution:

1. Identify the tags to style: `<h1>` for heading and `<p>` for paragraph.
2. Apply the `style` attribute with the `color` property directly inside the tags.

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="color: blue;">This is a Blue Heading</h1>
    <p style="color: red;">This is a red paragraph.</p>
</body>
</html>
```

Worked Example:

Embedding Internal Style Sheets Design an HTML page using an internal style sheet where the body has a light gray background and all `<h2>` elements have a green text color and are center-aligned.
Solution:

1. Add a `<style>` block within the `<head>` section of the HTML document.
2. Create a rule for `body` setting `background-color: lightgray;`.
3. Create a rule for `h2` setting `color: green;` and `text-align: center;`.

```
<!DOCTYPE html>
<html>
<head>
    <style>
```

```
        body {
            background-color: lightgray;
        }
        h2 {
            color: green;
            text-align: center;
        }
    </style>
</head>
<body>
    <h2>Green Centered Heading</h2>
    <p>This page has a light gray background.</p>
</body>
</html>
```

Worked Example:

Attaching External Style Sheets Explain how to attach an external CSS file named `styles.css` to an HTML document. The `styles.css` file should make all paragraphs bold and blue.
Solution:

1. Create the external CSS file `styles.css` containing the styling rules for the `p` selector.
2. Use the `<link>` tag within the `<head>` section of the HTML file to link the `styles.css` file.

Step 1: styles.css

```
p {
    font-weight: bold;
    color: blue;
}
```

Step 2: index.html

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <p>This paragraph is styled by an external CSS file.</p>
</body>
</html>
```

3.2 CSS Selectors

Methodology:

Common CSS Selectors Selectors are used to target specific HTML elements for styling.

1. **Element Selector**: Targets all instances of a specific HTML tag (e.g. `p` or `div`).
2. **Class Selector**: Targets elements with a specific class attribute. Preceded by a dot (`.classname`).
3. **ID Selector**: Targets a single unique element with a specific ID attribute. Preceded by a hash (`#idname`).
4. **Universal Selector**: Targets all elements on the page using an asterisk (`*`).

5. **Group Selector**: Targets multiple specific selectors by separating them with a comma (e.g. `h1 h2 p`).

Worked Example:

Using Class and ID Selectors Create an HTML document containing three paragraphs. Style the first paragraph using an ID to have a yellow background. Style the remaining two paragraphs using a class to have a dashed border.

Solution:

1. Assign an `id` to the first paragraph (e.g. `id="highlight"`).
2. Assign a `class` to the second and third paragraphs (e.g. `class="bordered"`).
3. Define the CSS rules using the `#highlight` and `.bordered` selectors inside a `<style>` block.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        #highlight {
            background-color: yellow;
        }
        .bordered {
            border: 2px dashed black;
            padding: 5px;
            margin: 5px 0;
        }
    </style>
</head>
<body>
    <p id="highlight">This is the highlighted paragraph.</p>
    <p class="bordered">This is the first bordered paragraph.</p>
    <p class="bordered">This is the second bordered paragraph.</p>
</body>
</html>
```

3.3 Formatting Text with CSS

Methodology:

Text Formatting Properties CSS provides several properties to control the appearance of text.

1. `color`: Sets the color of the text (e.g. `color: red;` or `color: #FF0000;`).
2. `text-align`: Sets the horizontal alignment (`left right center justify`).
3. `text-decoration`: Adds decorations to text (`none underline overline line-through`).
4. `text-transform`: Controls capitalization (`uppercase lowercase capitalize`).
5. `font-family`: Specifies the font type (e.g. `Arial sans-serif`).
6. `font-size`: Sets the size of the text (e.g. `16px 2em 120%`).
7. `font-weight`: Sets the boldness of the text (`normal bold bolder` or numeric values like `700`).

Worked Example:

Applying Text Formatting Create an HTML snippet that contains an `<h1>` and a `<p>`. Use internal CSS to style the `<h1>` to use Arial font uppercase text and an underline. Style the `<p>` to be italic justified and have a font size of 18px.

Solution:

1. Identify the target tags: `h1` and `p`.
2. Apply the text formatting properties in the `<style>` section mapping to the requirements.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        h1 {
            font-family: Arial, sans-serif;
            text-transform: uppercase;
            text-decoration: underline;
        }
        p {
            font-style: italic;
            text-align: justify;
            font-size: 18px;
        }
    </style>
</head>
<body>
    <h1>Web Design Fundamentals</h1>
    <p>
        Cascading Style Sheets (CSS) is a style sheet language used for
        describing the presentation of a document written in HTML. CSS
        is a cornerstone technology of the World Wide Web alongside
        HTML and JavaScript.
    </p>
</body>
</html>
```

3.4 Creating CSS Rules in Editors

Methodology:

Creating CSS Rules via IDE While hand-coding CSS is fundamental IDEs like Adobe Dreamweaver provide visual tools for defining CSS rules without writing code manually.

1. **Define a Selector**: Create a new selector (Class ID Tag or Compound) using the CSS Designer panel.
2. **Set Properties**: Adjust Layout Text Borders and Backgrounds using the visual property inspector interface.
3. **Apply to Elements**: Highlight the target HTML element in Design view and attach the newly created class or ID.
4. **Code Generation**: The IDE translates visual selections into correct CSS syntax inserting it into the appropriate stylesheet.

Worked Example:

Workflow for Creating a Rule Explain the step-by-step practical workflow of creating a class rule named `.highlight-box` that adds a blue border and yellow background and applying it to a `<div>`.
Solution:

1. **Step 1 (CSS Definition):** Declare the selector `.highlight-box` in the stylesheet section.
2. **Step 2 (Property Assignment):** Assign `border: 2px solid blue;` and `background-color: yellow;` to the selector.
3. **Step 3 (Application):** Add the attribute `class="highlight-box"` to the target `<div>` element in the HTML.

Resulting Code:

```
<style>
    .highlight-box {
        border: 2px solid blue;
        background-color: yellow;
    }
</style>

<div class="highlight-box">
    This box is styled using a custom CSS rule.
</div>
```

CHAPTER 4

Introduction to JavaScript

JavaScript is a versatile scripting language primarily used for adding interactivity and computational logic to web pages. In this chapter, we focus on the core computational elements of JavaScript. These include handling variables, applying arithmetic and logical operators, and utilizing built-in functions for robust problem solving.

4.1 Working with Variables and Built-in Functions

Variables store data that can be dynamically manipulated during program execution. To read and write data interactively in the browser, JavaScript provides straightforward dialog functions such as `prompt()` and `alert()`, as well as essential parsing functions to safely handle data types.

Methodology:

Capturing Input and Displaying Output To perform computations, we must accurately capture user input (which is typically read as a string) and convert it into numerical forms before evaluating expressions.

1. **Declare Variables:** Use `let` or `const` to define variables.
2. **Capture Input:** Use `prompt("Message")` to capture data from the user.
3. **Parse Input:** Use `parseInt(string)` to convert text into integers or `parseFloat(string)` to convert text into decimal-based numbers.
4. **Process Data:** Apply mathematical formulas and assignments.
5. **Display Output:** Show the processed result to the user utilizing the `alert("Message")` function.

Worked Example:

Calculating the Area of a Circle Write a JavaScript algorithm to calculate the area of a circle. Accept the radius from the user and display the final area.
Solution:

1. **Step 1:** Read the radius using `prompt()`.
2. **Step 2:** Convert the input string into a floating-point number using `parseFloat()`.
3. **Step 3:** Calculate the area using the formula Area $= \pi \times r^2$.
4. **Step 4:** Display the computed area using `alert()`.

JavaScript Code:

```
let radiusInput = prompt("Enter the radius of the circle:");
let radius = parseFloat(radiusInput);
let area = 3.14159 * radius * radius;
alert("The Area of the circle is:  " + area);
```

4.2 Arithmetic and Expression Evaluation

JavaScript natively supports standard mathematical operations. Furthermore, the `eval()` function is available to dynamically compute the result of mathematical expressions written as text strings, bypassing the need for manual parsing algorithms.

Methodology:

Dynamic Expression Evaluation using eval The `eval()` function calculates mathematical and logical string expressions directly.

1. Construct or receive a valid mathematical expression as a string (e.g. `"5 * 10 + 2"`).
2. Pass the string as an argument to `eval(expression)`.
3. The function parses the string, performs the calculations obeying standard operator precedence (BODMAS), and returns the evaluated numeric result.

Worked Example:

Evaluating a Dynamic Mathematical Expression Given a string variable containing an algebraic equation, compute the final numerical result using the `eval()` function.
Problem Statement: Evaluate the expression string `"(20 + 5) * 4 / 2 - 10"`.
Solution:

1. **Step 1:** Define the string expression.
2. **Step 2:** Pass the expression to `eval()`.
3. **Step 3:** Output the final computed integer.

JavaScript Code:

```
let expression = "(20 + 5) * 4 / 2 - 10";
let result = eval(expression);
alert("The evaluated result is:  " + result);
```

4.3 Comparison and Logical Operators

Comparison operators (such as `>`, `<`, `==`, `!=`) compare two operands and yield a boolean value (`true` or `false`). Logical operators (`&&` for AND, `||` for OR, `!` for NOT) combine multiple boolean expressions to formulate complex conditional constraints.

Methodology:

Logical Validation using Comparison Operators When validating constraints computationally, combining relational and logical operators allows testing multiple conditions efficiently.

1. Identify the individual logical requirements of the problem.
2. Connect conditions that must all be strictly met using the AND operator (`&&`).
3. Connect alternative conditions where at least one must be met using the OR operator (`||`).
4. Use the ternary operator (`condition ? trueResult : falseResult`) to branch the logic based on the boolean outcome.

Worked Example:

Determining Leap Year using Logical Expressions Formulate a logical expression to determine if a given year is a leap year. A year is considered a leap year if it is divisible by 400, OR if it is divisible by 4 AND NOT divisible by 100.
Solution:

1. **Step 1:** Prompt the user for the year and parse it using `parseInt()`.
2. **Step 2:** Check if the year is divisible by 400 using modulo (`year % 400 === 0`).
3. **Step 3:** Check if the year is divisible by 4 but not by 100 (`(year % 4 === 0) && (year % 100 !== 0)`).
4. **Step 4:** Combine the conditions using the OR operator (`||`).
5. **Step 5:** Use a ternary operator to output the final status.

JavaScript Code:

```
let year = parseInt(prompt("Enter a year:"));
let isLeap = (year % 400 === 0) || ((year % 4 === 0) && (year % 100 !== 0));
let result = isLeap ?  "Leap Year" :  "Not a Leap Year";
alert(year + " is " + result);
```

Worked Example:

Finding the Highest of Three Numbers Write a script to accept three numeric values from the user and find the largest number strictly using comparison and logical ternary operators.
Solution:

1. **Step 1:** Prompt the user for three distinct numbers and parse them via `parseFloat()`.
2. **Step 2:** Use logical AND (`&&`) combined with nested ternary operators to compare the numbers against each other.
3. **Step 3:** Display the highest number using an alert box.

JavaScript Code:

```
let n1 = parseFloat(prompt("Enter first number:"));
let n2 = parseFloat(prompt("Enter second number:"));
let n3 = parseFloat(prompt("Enter third number:"));

let max = (n1 >= n2 && n1 >= n3) ?  n1 :
          ((n2 >= n1 && n2 >= n3) ?  n2 :  n3);

alert("The highest number is:  " + max);
```

4.4 Complex Mathematical Modeling

Many web applications require robust handling of algebraic and scientific models. Built-in Math functions extend the standard arithmetic capabilities of JavaScript.

Methodology:

Solving Algebraic Formulas using the Math Object When working with roots and exponents, native operators can be paired with the built-in `Math` object to compute solutions cleanly.

1. Identify the variables required by the algebraic formula and capture them.
2. Substitute constants and variables into intermediate variables to simplify large equations.
3. Apply functions like `Math.sqrt(value)` to handle specific mathematical tasks like computing the square root.
4. Assemble the final equation logically to ensure correct precedence.

Worked Example:

Finding the Roots of a Quadratic Equation A quadratic equation is expressed as $ax^2 + bx + c = 0$. Write an algorithm to calculate the real roots of the equation utilizing standard arithmetic and the `Math.sqrt()` function.

Solution: The roots of the equation are calculated using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

1. **Step 1:** Read the mathematical coefficients a, b, and c.
2. **Step 2:** Compute the discriminant intermediate variable: $D = b^2 - 4ac$.
3. **Step 3:** Use `Math.sqrt(D)` to extract the square root of the discriminant.
4. **Step 4:** Calculate Root 1 as $\frac{-b+\sqrt{D}}{2a}$.
5. **Step 5:** Calculate Root 2 as $\frac{-b-\sqrt{D}}{2a}$.
6. **Step 6:** Formulate the output and display both roots.

JavaScript Code:

```
let a = parseFloat(prompt("Enter coefficient a:"));
let b = parseFloat(prompt("Enter coefficient b:"));
let c = parseFloat(prompt("Enter coefficient c:"));

let D = (b * b) - (4 * a * c);

let root1 = (-b + Math.sqrt(D)) / (2 * a);
let root2 = (-b - Math.sqrt(D)) / (2 * a);

alert("Root 1:  " + root1 + " Root 2:  " + root2);
```

CHAPTER 5

User Defined Function and Decision Making in JavaScript

5.1 Conditional Statements

Conditional statements are used to perform different actions based on different conditions. JavaScript supports `if`, `if...else`, `if...else if...else`, and `switch` statements for decision-making.

Methodology:

Implementing Conditional Statements **1. The `if...else` Statement:** Evaluates a condition and executes a block of code if true, and another block if false.

```
if (condition) {
    // block of code to be executed if the condition is true
} else {
    // block of code to be executed if the condition is false
}
```

2. The `switch` Statement: Evaluates an expression and matches its value to a `case` label. When a match is found, the associated block of code is executed.

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Worked Example:

Check whether given character is a vowel or consonant using switch case **Problem:** Write a JavaScript program to check if a given character is a vowel or a consonant.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Vowel or Consonant</title>
</head>
<body>
    <h2>Check Vowel or Consonant</h2>
```

```
    <script>
        let char = 'a'; // You can change this character to test
        let result = "";

        // Convert to lowercase to handle uppercase inputs
        switch(char.toLowerCase()) {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
                result = char + " is a Vowel.";
                break;
            default:
                result = char + " is a Consonant.";
        }
        document.write("<h3>" + result + "</h3>");
    </script>
</body>
</html>
```

5.2 Loop Statements

Loops are used to execute a block of code repeatedly as long as a specified condition is true. The common loop structures in JavaScript are `for`, `while`, and `do...while`.

Methodology:

Utilizing Loops **1. The `for` Loop:** Ideal when you know exactly how many times you want to loop through a block of code.

```
for (initialization; condition; increment/decrement) {
  // code block to be executed
}
```

2. The `while` Loop: Loops through a block of code as long as a specified condition is true.

```
while (condition) {
  // code block to be executed
}
```

3. The `do...while` Loop: A variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {
  // code block to be executed
} while (condition);
```

Worked Example:

Print first 10 even numbers **Problem:** Write a JavaScript program to print the first 10 even numbers.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>First 10 Even Numbers</title>
```

```
</head>
<body>
    <h2>First 10 Even Numbers:</h2>
    <script>
        let count = 0;
        let num = 2; // Starting from the first positive even number

        while (count < 10) {
            document.write(num + "<br>");
            num += 2;
            count++;
        }
    </script>
</body>
</html>
```

Worked Example:

Calculate power of a given number **Problem:** Write a JavaScript program to calculate the power of a given number (x^y).
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Calculate Power</title>
</head>
<body>
    <h2>Power Calculator</h2>
    <script>
        let base = 5;
        let exponent = 3;
        let result = 1;

        for (let i = 1; i <= exponent; i++) {
            result *= base;
        }

        document.write("<h3>" + base + " raised to the power of "
                        + exponent + " is: " + result + "</h3>");
    </script>
</body>
</html>
```

Worked Example:

Print multiplication table of a given number **Problem:** Write a JavaScript program to print the multiplication table of a given number.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Multiplication Table</title>
</head>
<body>
```

```
    <h2>Multiplication Table of 7</h2>
    <script>
        let number = 7;

        document.write("<table border='1' cellpadding='5'>");
        for (let i = 1; i <= 10; i++) {
            document.write("<tr>");
            document.write("<td>" + number + " x " + i + "</td>");
            document.write("<td> = </td>");
            document.write("<td>" + (number * i) + "</td>");
            document.write("</tr>");
        }
        document.write("</table>");
    </script>
</body>
</html>
```

5.3 User Defined Functions

A JavaScript function is a block of code designed to perform a particular task. A function is executed when it is invoked or called.

Methodology:

Declaring and Calling User Defined Functions **1. Function Declaration:** A function is defined with the `function` keyword, followed by a name, followed by parentheses `()`. The code to be executed by the function is placed inside curly brackets `{}`.

```
function functionName(parameter1, parameter2, ...) {
  // code to be executed
  return result; // optional return statement
}
```

2. Function Invocation: To execute the function, you call it by its name followed by parentheses.

```
functionName(value1, value2);
```

Worked Example:

Calculate the factorial of a given number **Problem:** Write a JavaScript function to calculate the factorial of a given number.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Factorial Calculation</title>
    <script>
        // User-defined function to calculate factorial
        function calculateFactorial(n) {
            if (n === 0 || n === 1) {
                return 1;
            }
            let fact = 1;
            for (let i = 2; i <= n; i++) {
                fact *= i;
```

```
            }
            return fact;
        }
    </script>
</head>
<body>
    <h2>Factorial of 5</h2>
    <script>
        let num = 5;
        let result = calculateFactorial(num);
        document.write("The factorial of " + num + " is " + result);
    </script>
</body>
</html>
```

5.4 Document Object Model (DOM) and HTML Events

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. JavaScript interacts with the DOM to access and manipulate HTML elements (Window, Document, Form, Input elements).

Events are actions that happen in the system, which the system tells you about so you can respond to them (e.g., `onchange`, `onclick`, `onmouseover`, `onmouseout`, `onkeydown`, `onload`).

Methodology:

Accessing Elements and Handling Events **1. Accessing HTML Elements:** You can access HTML elements using methods provided by the `document` object.

- `document.getElementById(id)`: Finds an element by its ID.
- `document.getElementsByName(name)`: Finds elements by their name attribute.

2. Handling Events: You can attach event listeners to HTML elements to execute JavaScript code when a specific event occurs.

```
<button onclick="myFunction()">Click Me</button>
```

Worked Example:

Display student name and address in a dialog box **Problem:** Take input of a student's name and address and display it in an alert dialog box.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Information</title>
    <script>
        function displayInfo() {
            // Accessing form elements using DOM
            let name = document.getElementById("studentName").value;
            let address = document.getElementById("studentAddress").value;

            // Displaying information using alert dialog
            alert("Student Name: " + name + "\nAddress: " + address);
        }
```

```
    </script>
</head>
<body>
    <h2>Enter Student Details</h2>
    <form>
        Name: <input type="text" id="studentName"><br><br>
        Address: <textarea id="studentAddress"></textarea><br><br>
        <input type="button" value="Submit" onclick="displayInfo()">
    </form>
</body>
</html>
```

Worked Example:

Change background color using a combo box **Problem:** Change the background color of the webpage as selected by the user from a list of colors given in a combo box (drop-down list).
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Change Background Color</title>
    <script>
        function changeColor() {
            // Get the selected color from the combo box
            let colorSelect = document.getElementById("colorList");
            let selectedColor = colorSelect.value;

            // Change the document background color
            document.body.style.backgroundColor = selectedColor;
        }
    </script>
</head>
<body>
    <h2>Select Background Color</h2>
    <select id="colorList" onchange="changeColor()">
        <option value="white">White</option>
        <option value="red">Red</option>
        <option value="green">Green</option>
        <option value="blue">Blue</option>
        <option value="yellow">Yellow</option>
    </select>
</body>
</html>
```

Worked Example:

Perform arithmetic operations using Radio buttons **Problem:** Perform arithmetic operations on two numbers entered into textboxes. Use Radio buttons to select arithmetic operations (Addition, Subtraction, Multiplication, and Division). Display the result in another textbox.
Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Arithmetic Operations</title>
```

```
    <script>
        function calculate() {
            let num1 = parseFloat(document.getElementById("n1").value);
            let num2 = parseFloat(document.getElementById("n2").value);
            let result = 0;

            // Check which radio button is selected
            if (document.getElementById("add").checked) {
                result = num1 + num2;
            } else if (document.getElementById("sub").checked) {
                result = num1 - num2;
            } else if (document.getElementById("mul").checked) {
                result = num1 * num2;
            } else if (document.getElementById("div").checked) {
                if(num2 !== 0) {
                    result = num1 / num2;
                } else {
                    result = "Cannot divide by zero";
                }
            }

            // Display the result
            document.getElementById("ans").value = result;
        }
    </script>
</head>
<body>
    <h2>Arithmetic Calculator</h2>
    Number 1: <input type="text" id="n1"><br><br>
    Number 2: <input type="text" id="n2"><br><br>

    Operations:<br>
    <input type="radio" name="operation" id="add" onclick="calculate()"> Addition<br>
    <input type="radio" name="operation" id="sub" onclick="calculate()"> Subtraction<br>
    <input type="radio" name="operation" id="mul" onclick="calculate()"> Multiplication<br>
    <input type="radio" name="operation" id="div" onclick="calculate()"> Division<br><br>

    Result: <input type="text" id="ans" readonly>
</body>
</html>
```

Worked Example:

Calculate factorial using textboxes **Problem:** Calculate the factorial of a given number entered into a textbox and display the result in another textbox.

Solution:

```
<!DOCTYPE html>
<html>
<head>
    <title>Factorial with DOM</title>
    <script>
        function getFactorial() {
            let n = parseInt(document.getElementById("num").value);
            let fact = 1;
```

```
            if (isNaN(n)) {
                fact = "Invalid Input";
            } else if (n < 0) {
                fact = "Undefined for negative numbers";
            } else {
                for (let i = 1; i <= n; i++) {
                    fact *= i;
                }
            }

            document.getElementById("resultFact").value = fact;
        }
    </script>
</head>
<body>
    <h2>Factorial Calculator</h2>
    Enter a number: <input type="text" id="num"><br><br>
    <input type="button" value="Calculate Factorial" onclick="getFactorial()"><br><br>
    Result: <input type="text" id="resultFact" readonly>
</body>
</html>
```