Web Development Fundamentals: HTML, CSS, and JavaScript

A comprehensive guide to the building blocks of modern web development

Course Agenda

01

HTML Fundamentals

Structure, tags, and content organization

02

CSS Styling

Selectors, properties, and layout techniques

03

JavaScript Functionality

DOM manipulation, events, and logic

04

Integration

Combining technologies for dynamic websites

Throughout this course, we'll explore how these technologies work together to create responsive, interactive web experiences.

HTML: The Structure

Key components include:

HTML (HyperText Markup Language) provides the fundamental structure for web pages. It organises content using a system of elements and tags.

- Document declaration: `<!DOCTYPE html>`
- Head section: `<head>` for metadata
- Body section: `<body>` for visible content
- Content containers: `<div>`, `<section>`
- User input elements: `<input>`, `<button>`

HTML: Tags and Attributes

Structure Tags

- `<html>`: Root element
- `<head>`: Document metadata
- `<body>`: Visible content
- `<div>`: Generic container

Metadata Tags

- `<title>`: Page title
- `<meta>`: Additional information
- `<link>`: External resources

Attributes

- `id`: Unique identifier
- `class`: Grouping elements
- `placeholder`: Input field hint

HTML elements can be targeted via their IDs and classes for styling with CSS and manipulation with JavaScript.

CSS: Styling Your Content

Cascading Style Sheets (CSS) control the visual presentation of HTML elements. CSS transforms plain HTML into visually appealing interfaces.

CSS can be linked to HTML using the `<link>` tag:

```
<link rel="stylesheet" href="styles.css">
```

CSS consists of:

- **Selectors**: Target HTML elements
- **Properties**: Define what to change
- **Values**: Specify how to change it

CSS: Selectors and Properties

Selectors

- Element: `div { }`
- Class: `.chat-box { }`
- ID: `#message-input { }`
- Pseudo-class: `button:hover { }`

Common Properties

- `font-family`, `font-size`
- `background-color`
- `border-radius`, `box-shadow`
- `padding`, `margin`

Units & Values

- Pixels: `px`
- Viewport height: `vh`
- Hex colours: `#835E54`
- RGBA: `rgba(201,144,124,0.5)`

CSS properties control everything from text appearance to spacing, borders, and colours. The box model (content, padding, border, margin) is fundamental to understanding layout.

CSS: Modern Layout Techniques

Flexbox Layout

Flexbox provides a more efficient way to arrange elements, even when their size is unknown or dynamic.

```
.container {
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: center;
}
```

Key properties include:

- `display: flex`
- `flex-direction`
- `justify-content`
- `align-items`

Flexbox simplifies complex layouts that would be difficult with traditional CSS positioning methods. It's particularly useful for responsive designs.

JavaScript: Adding Interactivity

JavaScript brings web pages to life by enabling dynamic content and user interactions. It can be included in HTML using the `<script>` tag:

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

JavaScript allows you to:

- Manipulate the Document Object Model (DOM)
- Respond to user events (clicks, keypresses)
- Create and modify content dynamically
- Control page behaviour and appearance

JavaScript is what transforms static pages into interactive applications.

JavaScript: DOM Manipulation

Select Elements

```
const chatBox = document.getElementById('chat-box');
```

Create Elements

```
const message = document.createElement('div');
```

Modify Elements

```
message.className = 'message';
message.textContent = text;
```

Append Elements

```
chatBox.appendChild(message);
```

The Document Object Model (DOM) represents the page as a tree of objects that JavaScript can manipulate. Through DOM manipulation, you can dynamically update content without reloading the page.

JavaScript: Events and Listeners

Event listeners allow JavaScript to respond to user actions. They're crucial for creating interactive experiences.

```
// Click event listener
sendButton.addEventListener('click', function() {
  sendMessage();
});

// Keypress event listener
input.addEventListener('keypress', function(e) {
  if (e.key === 'Enter') {
    sendMessage();
  }
});
```

Common Events

- `'click'`: Mouse clicks
- `'keypress'`: Keyboard input
- `'submit'`: Form submission
- `'load'`: Page loading
- `'scroll'`: User scrolling
- `'mouseover'`: Hover actions

JavaScript: Functions and Logic

Function Declaration

```
function sendMessage() {
  // Function code here
}
```

Traditional way to define functions

Arrow Functions

```
const sendMessage = () => {
  // Function code here
}
```

Modern, concise syntax introduced in ES6

Conditional Logic

```
if (messageText.trim() !== '') {
  addMessage(messageText);
} else {
  // Handle empty message
}
```

Control flow with if/else statements

Functions encapsulate reusable code blocks, while conditional logic allows for decision-making based on different scenarios and user inputs.

JavaScript: Timers and String Manipulation

Timers

JavaScript can execute code after a delay or at regular intervals:

```
// Execute once after 2 seconds
setTimeout(() => {
  scrollToBottom();
}, 2000);

// Execute repeatedly every 1 second
setInterval(() => {
  updateTime();
}, 1000);
```

Timers are useful for animations, updates, and delayed actions.

String Manipulation

JavaScript offers powerful string handling capabilities:

```
// Remove whitespace
const text = input.value.trim();

// Template literals
const greeting = `Hello, ${username}!`;

// String concatenation
const fullName = firstName + ' ' + lastName;
```

String manipulation is essential for processing user input and generating dynamic content.

JavaScript: Async/Await for API Calls

`async/await` provides a cleaner, more readable way to handle asynchronous operations, making promise-based code appear synchronous. It simplifies API requests by replacing complex `.then()` chains with sequential-looking code and allowing for easy error handling with `try/catch` blocks.

GET Request Example

```
async function fetchData() {
  try {
    // Await the response from the API
    const response = await fetch('https://api.example.com/users');

    // Check if the response was successful
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    // Await parsing the JSON data
    const data = await response.json();
    console.log('Fetched data:', data);

  } catch (error) {
    // Catch any errors during fetch or parsing
    console.error('Error fetching data:', error);
  }
}
fetchData(); // Call the async function
```

This example demonstrates fetching data from an endpoint. `await` pauses the function execution until the promise settles, making the code flow top-to-bottom. Errors are gracefully handled by the `try/catch` block.

POST Request Example

```
async function createItem(itemData) {
  try {
    const response = await fetch('https://api.example.com/items', {
      method: 'POST', // Specify the HTTP method
      headers: {
        'Content-Type': 'application/json', // Content type header
        'Accept': 'application/json' // Expected response type
      },
      body: JSON.stringify(itemData) // Convert data to JSON string
    });

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

    const newItem = await response.json();
    console.log('Item created:', newItem);

  } catch (error) {
    console.error('Error creating item:', error);
  }
}
// Example usage:
createItem({ name: 'New Widget', price: 29.99 });
```

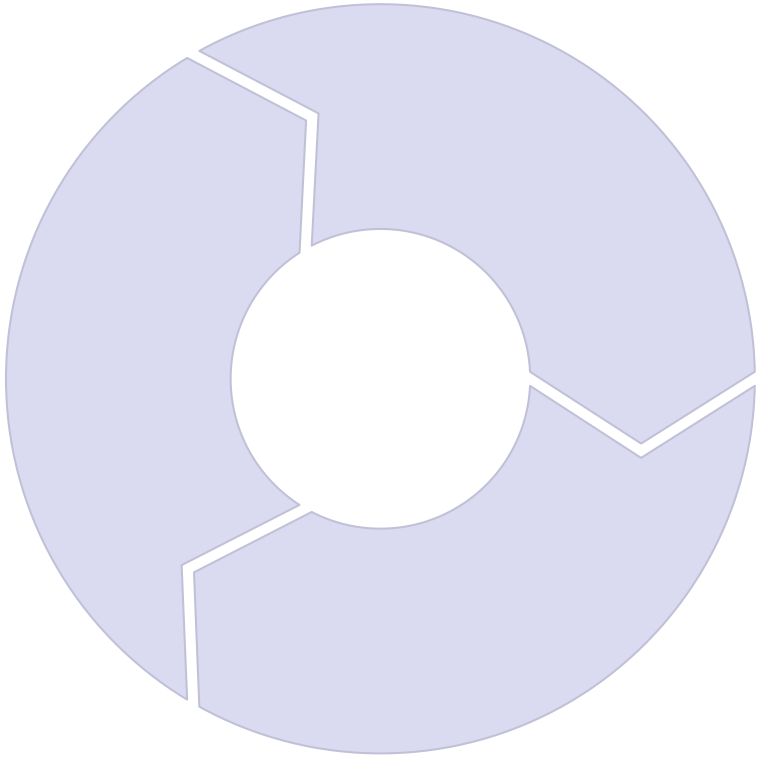
For POST requests, `async/await` works similarly. We include `method`, `headers`, and a `body` (often stringified JSON) in the fetch options. This allows sending data to the server securely.

Integration: Bringing It All Together

The true power of web development comes from the integration of HTML, CSS, and JavaScript:

HTML

Provides structure and content with elements that have IDs and classes



CSS

Styles those elements using selectors that target the IDs and classes

JavaScript

Manipulates the elements and applies dynamic behaviour

This integration creates a seamless user experience where content, presentation, and behaviour work in harmony.

Debugging Basics

Browser Developer Tools

Modern browsers include powerful tools for debugging web applications:

- **Elements panel**: Inspect and modify HTML/CSS
- **Console**: View JavaScript errors and logs
- **Network tab**: Monitor resource loading
- **Sources panel**: Debug JavaScript code

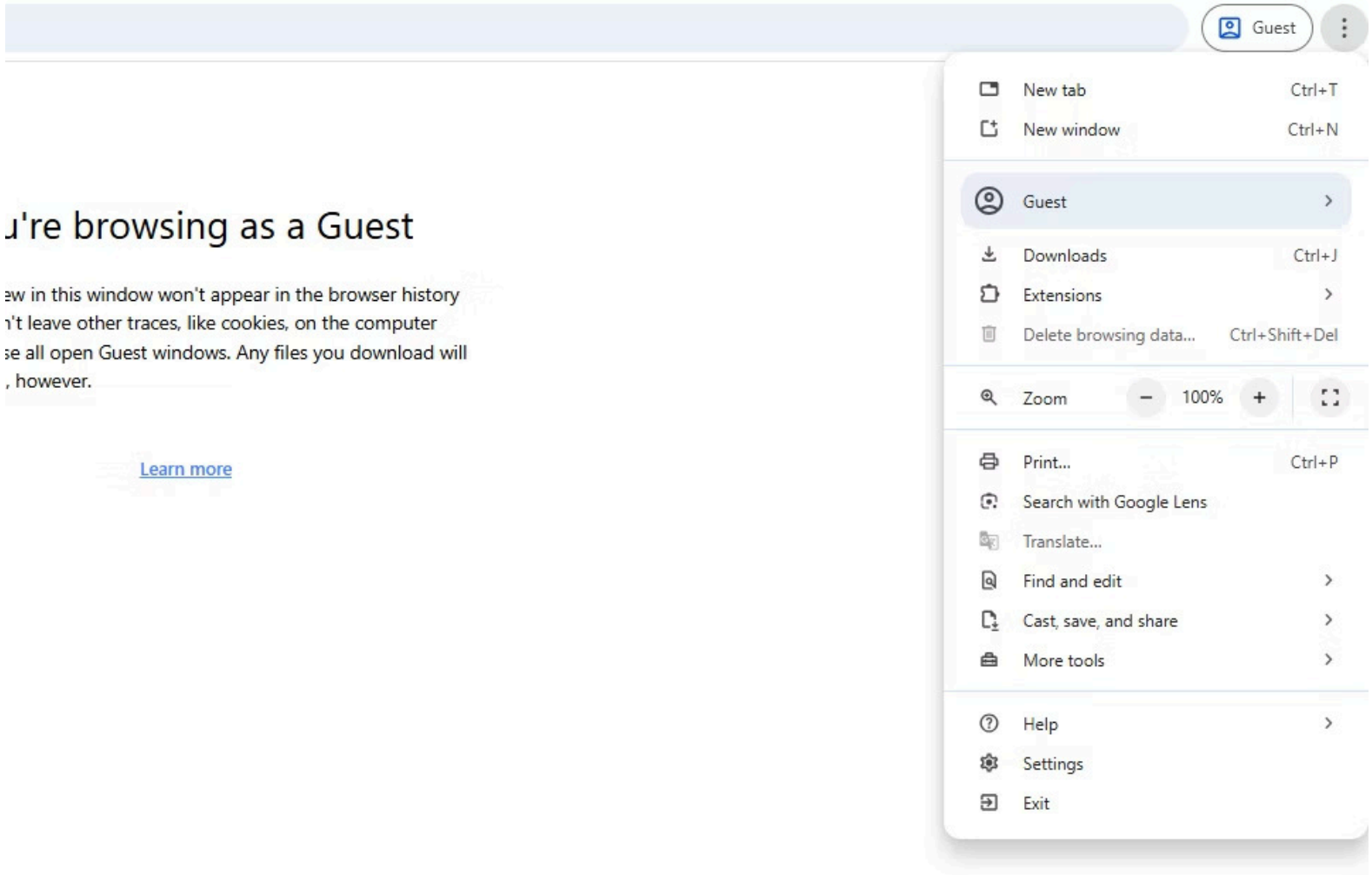
Access developer tools by right-clicking and selecting "Inspect" or pressing F12.

The console is particularly useful for JavaScript debugging:

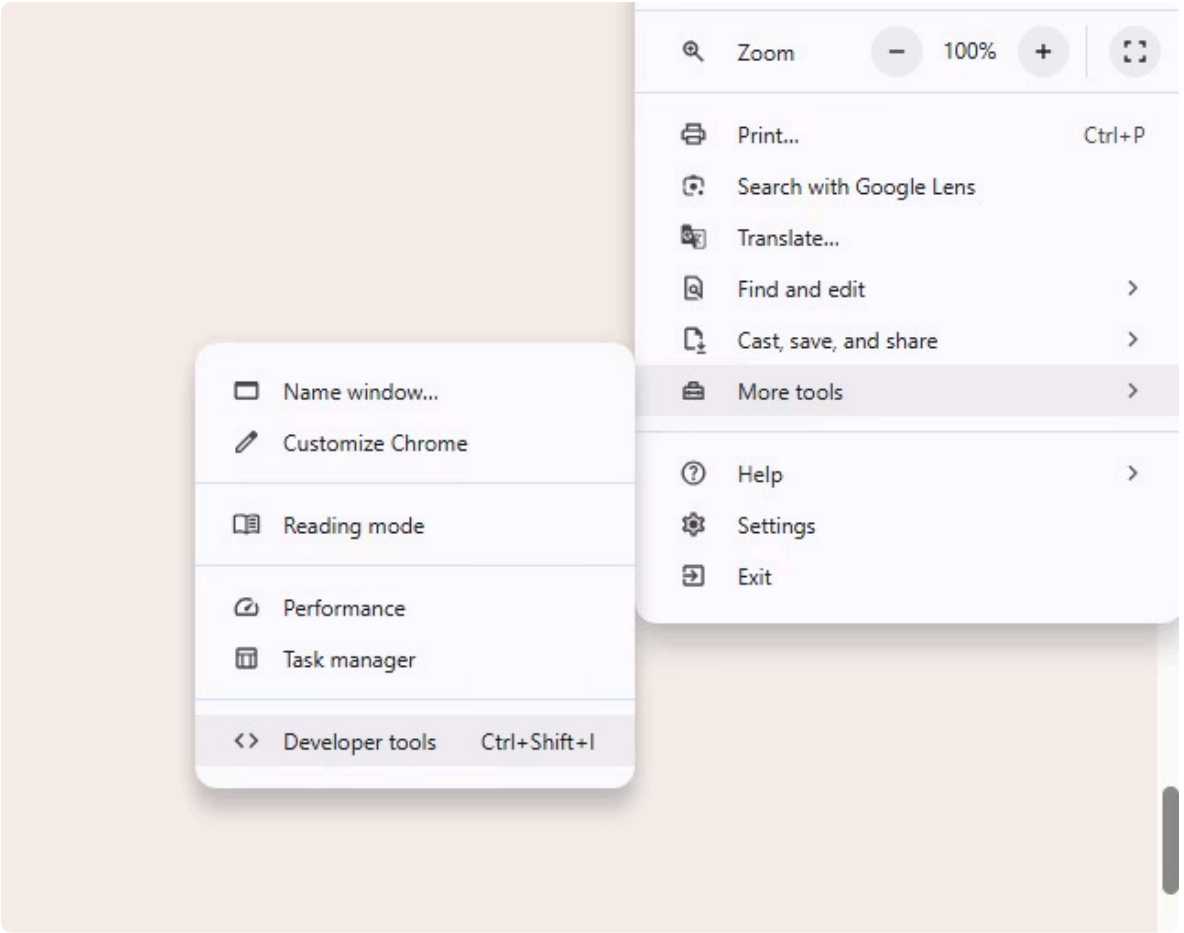
```
// Output values for debugging
console.log('Message sent:', messageText);

// Show errors
console.error('Connection failed');
```

Select "More Tools"



Select Developer tools



DevTools - gamma.app/docs/Week-02-03-Web-Development-Fundamentals-HTML-CSS-and-JavaScript-fwq5hkmvuczq738?mode=doc

Elements | Console | Sources | Network | Performance | Memory | Application | Privacy and security | Lighthou

```
<!DOCTYPE html>
<html lang="zh-tw" translate="no" class="notranslate" data-app-page="true" data-theme="
t" dir="ltr" style="--100vh: 100vh; color-scheme: light;">
  ▸ <head> ⋯ </head>
⋯ ▾ <body class="chakra-ui-light"> == $0
    ▸ <div id="__next">⋯</div>
    ▸ <script id="__NEXT_DATA__" type="application/json">⋯</script>
      <pre id="__textLabelMeasure" tabindex="-1" style="white-space: pre; width: auto; border:
      1px solid transparent; padding: 4px; margin: 0px; letter-spacing: -0.03em; opacity: 0; p
      osition: absolute; top: -500px; left: 0px; z-index: 9999; pointer-events: none; user-sel
      ect: none; alignment-baseline: mathematical; dominant-baseline: mathematical; line-heigh
      t: 1.3;"></pre>
      <pre id="__textMeasure" tabindex="-1" style="white-space: pre; width: auto; border: 1px
      solid transparent; padding: 4px; margin: 0px; letter-spacing: -0.03em; opacity: 0; posit
      ion: absolute; top: -500px; left: 0px; z-index: 9999; pointer-events: none; user-select:
      none; alignment-baseline: mathematical; dominant-baseline: mathematical;"></pre>
      <div class="gamma-tooltip-portal"></div>
    ▸ <script type="text/javascript" async>⋯</script>
    ▸ <script type="text/javascript" async>⋯</script>
    ▸ <script type="text/javascript" async>⋯</script>
    ▸ <next-route-announcer>⋯</next-route-announcer>
    ▸ <div class="chakra-portal">⋯</div>
    ▸ <iframe src="https://a187838077.cdn.optimizely.com/client_storage/a187838077.html"
      hidden tabindex="-1" title="Optimizely Internal Frame" height="0" width="0" style="disp
      lay: none;">⋯</iframe>
    ▸ <ck-widget data-powered-by="Churnkey.co Cancel Flows" vce-ready>⋯</ck-widget>
    ▸ <ck-pause-wall data-powered-by="Churnkey.co Cancel Flows" vce-ready>⋯</ck-pause-wall>
    ▸ <ck-failed-payment-wall data-powered-by="Churnkey.co Automated Payment Recovery" vce-
      ready>⋯</ck-failed-payment-wall>
    ▸ <ck-managed-flow data-powered-by="Churnkey.co Cancel Flows" vce-ready>⋯</ck-managed-
```

Styles | Computed

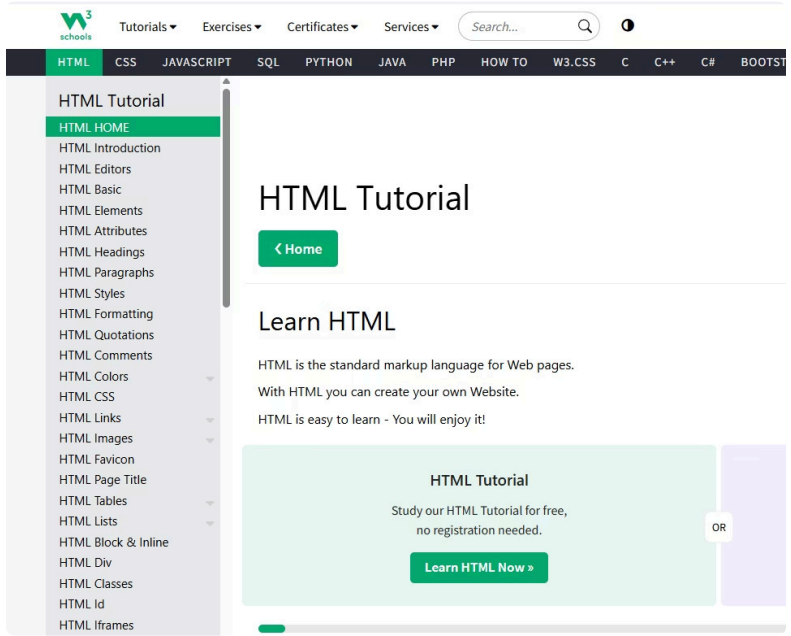
Filter

```
▸ background-attach
▸ background-clip
▸ background-color
▸ background-image
▸ background-origin
▸ background-positi
▸ background-positi
▸ background-repeat
▸ background-size
▸ border-bottom-col
▸ border-bottom-sty
▸ border-bottom-wid
▸ border-left-color
▸ border-left-style
▸ border-left-width
```

www.w3schools.com

W3Schools.com

W3Schools offers free online tutorials, references and exercises in all the major languages of the web. Covering popular subjects like HTML, CSS, JavaScript, Python, SQL, Java, and many, many...



www.w3schools.com