### **What is HTML5?**

### **HTML5 is the fifth version of the HyperText Markup Language, designed to enhance the structure and presentation of content on the web. It introduced a plethora of new elements, attributes, APIs, and features, making it a robust tool for creating interactive and dynamic web applications.**

### **New HTML5 Elements**

### **HTML5 introduced several semantic elements that improve the structure and readability of your code.**

### **Intermediate Level:**

### **Semantic elements: <header>, <footer>, <nav>, <section>, <article>, <aside>, <figure>, <figcaption>, <details>, <summary>. These elements provide better structure and meaning to your content, improving SEO and accessibility.**

### **Form elements: <datalist>, <output>, <progress>, <meter>, <keygen>. These elements enhance form functionality and user experience.**

### **Multimedia elements: <audio>, <video>. Embed audio and video directly into your web pages without relying on plugins.**

### **Advanced Level:**

### **Custom data attributes: data-\* attributes allow you to store custom data within HTML elements, providing a way to store additional information without modifying the DOM.**

### **Canvas: <canvas> element provides a canvas for dynamic, scriptable rendering of graphics. It's used for creating animations, games, and data visualizations.**

### **Web Workers: Web Workers enable JavaScript code to run in the background without interfering with the user interface, improving performance and responsiveness.**

### **New JavaScript APIs**

### **HTML5 introduced several JavaScript APIs to enhance web application development.**

### **Intermediate Level:**

### **Geolocation API: Access the user's geographical location.**

### **Storage APIs: localStorage and sessionStorage for storing data locally.**

### **Drag and Drop API: Enable drag-and-drop functionality.**

### **Application Cache API: Allow web applications to work offline.**

### **Advanced Level:**

### **WebSocket API: Enable real-time communication between the browser and server.**

### **File API: Access files on the user's computer.**

### **Server-Sent Events: Receive updates from the server without polling.**

### **IndexedDB: Create client-side databases.**

### **When is HTML5 Fully Compliant with All Browsers?**

### **HTML5 has been adopted by major browsers for several years. However, full compliance with all features can vary across browsers and their versions. It's essential to test your code across different platforms to ensure compatibility.**

### **Markup Example Before and After HTML5**

### **Before HTML5:**

### **HTML**

### **<div id="header">**

### **<h1>My Website</h1>**

### **<ul>**

### **<li><a href="#">Home</a></li>**

### **<li><a href="#">About</a></li>**

### **<li><a href="#">Contact</a></li>**

### **</ul>**

### **</div>**

### 

### **Use code with caution.**

### **After HTML5:**

### **HTML**

### **<header>**

### **<h1>My Website</h1>**

### **<nav>**

### **<ul>**

### **<li><a href="#">Home</a></li>**

### **<li><a href="#">About</a></li>**

### **<li><a href="#">Contact</a></li>**

### **</ul>**

### **</nav>**

### **</header>**

### 

### **Use code with caution.**

### **The HTML5 version uses semantic elements like <header> and <nav> for better structure and meaning.**

### **Detecting HTML5 Features**

### **To detect HTML5 features, you can use feature detection:**

### **JavaScript**

### **if (typeof(Storage) !== "undefined") {**

### **// localStorage and sessionStorage are supported**

### **} else {**

### **// No Web Storage support**

### **}**

### 

### **Use code with caution.**

### **Intermediate and Advanced Level:**

### **For more complex feature detection, consider using libraries like Modernizr or creating custom feature detection logic.**

### **Additional Considerations:**

### **Progressive Enhancement: Build your website to work without JavaScript and then enhance it with JavaScript for added functionality.**

### **Performance Optimization: Optimize your HTML5 code for performance, especially when using complex features like canvas or Web Workers.**

### **Accessibility: Ensure your HTML5 content is accessible to users with disabilities by following accessibility guidelines.**

### **By understanding these concepts and examples, you can effectively leverage HTML5 to create modern, interactive, and accessible web applications.**

### **Would you like to delve deeper into a specific aspect of HTML5?**

### 

### **These apis have interfaces, events and properties**

### 

### **Geolocation API**

**Intermediate Example:** Weather App

* **Functionality:** Fetch user's location, then display the current weather based on that location using a weather API.

**Code Snippet:** JavaScript  
function getLocation() {

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(success, error);

} else {

alert("Geolocation is not supported by this browser.");

}

}

function success(position) {

const lat = position.coords.latitude;

const lon = position.coords.longitude;

// Fetch weather data using a weather API with lat and lon

}

function error() {

alert("Geolocation failed.");

}

* Use code with caution.

The developer can now access this location information in a couple of different ways:

* [Geolocation.getCurrentPosition()](https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/getCurrentPosition): Retrieves the device's current location.
* [Geolocation.watchPosition()](https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/watchPosition): Registers a handler function that will be called automatically each time the position of the device changes, returning the updated location.

### **Storage APIs**

**Intermediate Example:** User Preferences

* **Functionality:** Store user preferences like theme, font size, and notification settings using localStorage.

**Code Snippet:** JavaScript  
function savePreferences() {

const theme = document.getElementById('theme').value;

const fontSize = document.getElementById('fontSize').value;

const notifications = document.getElementById('notifications').checked;

localStorage.setItem('theme', theme);

localStorage.setItem('fontSize', fontSize);

localStorage.setItem('notifications', notifications);

}

function loadPreferences() {

const savedTheme = localStorage.getItem('theme');

const savedFontSize = localStorage.getItem('fontSize');

const savedNotifications = localStorage.getItem('notifications');

// Apply saved preferences to UI elements

}

* Use code with caution.

The two mechanisms within Web Storage are as follows:

* sessionStorage maintains a separate storage area for each given [origin](https://developer.mozilla.org/en-US/docs/Glossary/Origin) that's available for the duration of the page session (as long as the browser tab is open, including page reloads and restores).
* localStorage does the same thing, but persists even when the browser is closed and reopened.

Both sessionStorage and localStorage in Web Storage are synchronous in nature. This means that when data is set, retrieved, or removed from these storage mechanisms, the operations are performed synchronously, blocking the execution of other JavaScript code until the operation is completed. This synchronous behavior can potentially affect the performance of the web application, especially if there is a large amount of data being stored or retrieved.

### **Drag and Drop API**

**Intermediate Example:** File Uploader

* **Functionality:** Allow users to drag and drop files onto a designated area for uploading.

**Code Snippet:** JavaScript  
const dropArea = document.getElementById('drop-area');

dropArea.addEventListener('dragover', (e) => {

e.preventDefault();

dropArea.classList.add('hover');

});

dropArea.addEventListener('dragleave', () => {

dropArea.classList.remove('hover');

});

dropArea.addEventListener('drop', (e) => {

e.preventDefault();

dropArea.classList.remove('hover');

const files = e.dataTransfer.files;

// Process uploaded files

});

* Use code with caution.

HTML drag-and-drop uses the [DOM event model](https://developer.mozilla.org/en-US/docs/Web/API/Event) and [*drag events*](https://developer.mozilla.org/en-US/docs/Web/API/DragEvent) inherited from [mouse events](https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent). A typical *drag operation* begins when a user selects a *draggable* element, continues when the user drags the element to a *droppable* element, and then ends when the user releases the dragged element.

Each [drag event type](https://developer.mozilla.org/en-US/docs/Web/API/DragEvent#event_types) has an associated event handler:

| **Event** | **Fires when...** |
| --- | --- |
| [drag](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/drag_event) | ...a *dragged item* (element or text selection) is dragged. |
| [dragend](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dragend_event) | ...a drag operation ends (such as releasing a mouse button or hitting the Esc key; see [Finishing a Drag](https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/Drag_operations#finishing_a_drag).) |
| [dragenter](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dragenter_event) | ...a dragged item enters a valid drop target. (See [Specifying Drop Targets](https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/Drag_operations#specifying_drop_targets).) |
| [dragleave](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dragleave_event) | ...a dragged item leaves a valid drop target. |
| [dragover](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dragover_event) | ...a dragged item is being dragged over a valid drop target, every few hundred milliseconds. |
| [dragstart](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dragstart_event) | ...the user starts dragging an item. (See [Starting a Drag Operation](https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/Drag_operations#starting_a_drag_operation).) |
| [drop](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/drop_event) | ...an item is dropped on a valid drop target. (See [Performing a Drop](https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/Drag_operations#performing_a_drop).) |

### **Application Cache API**

**Intermediate Example:** Offline Web App

* **Functionality:** Cache essential resources for an app to work offline.

**Code Snippet:** HTML  
<html manifest="app.manifest">

* Use code with caution.  
   **app.manifest** file would list cached resources.

**Create the manifest file (cache.appcache):**

CACHE MANIFEST

# Version 1.0

CACHE:

index.html

style.css

script.js

image.png

NETWORK:

\*

1. The first time the page loads, the browser downloads the resources listed in the manifest file and stores them in the application cache.
2. Subsequent visits, even offline, will use the cached resources.
3. To update the cache, change the version number in the manifest file.

**Important Notes:**

* The manifest file must be a text file with the .appcache extension.
* The MIME type for the manifest file should be text/cache-manifest.
* The browser will only update the cache if the manifest file has changed.
* There are events like updateready, noupdate, error, etc., which can be used to monitor the cache status.

**Additional Considerations:**

* For more complex scenarios, consider using Service Workers, which offer more flexibility and control.
* Always test your application thoroughly in offline mode.
* Be aware of the limitations of Application Cache API.

## **Advanced Level**

### **WebSocket API**

**Advanced Example:** Real-time Chat Application

* **Functionality:** Create a chat application where messages are sent and received instantly without page refreshes.

**Code Snippet:** JavaScript  
const socket = new WebSocket('ws://your-websocket-server');

socket.onopen = function(event) {

console.log('WebSocket connected');

};

socket.onmessage = function(event) {

const message = JSON.parse(event.data);

// Display received message

};

socket.onclose = function(event) {

console.log('WebSocket disconnected');

};

The WebSocket API is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

* [AsyncAPI](https://www.asyncapi.com/): A specification for describing event-driven architectures based on protocols like WebSocket. You can use it to describe WebSocket-based APIs just as you would describe REST APIs with the OpenAPI specification. Learn [why you should consider using AsyncAPI with WebSocket](https://www.asyncapi.com/blog/websocket-part1) and [how to do so](https://www.asyncapi.com/blog/websocket-part2).
* [HumbleNet](https://hacks.mozilla.org/2017/06/introducing-humblenet-a-cross-platform-networking-library-that-works-in-the-browser/): A cross-platform networking library that works in the browser. It consists of a C wrapper around WebSockets and WebRTC that abstracts away cross-browser differences, facilitating the creation of multi-user networking functionality for games and other apps.
* [µWebSockets](https://github.com/uNetworking/uWebSockets): Highly scalable WebSocket server and client implementation for [C++11](https://isocpp.org/) and [Node.js](https://nodejs.org/).
* [Socket.IO](https://socket.io/): A long polling/WebSocket based third party transfer protocol for [Node.js](https://nodejs.org/).
* [SocketCluster](https://socketcluster.io/): A pub/sub WebSocket framework for [Node.js](https://nodejs.org/) with a focus on scalability.
* [WebSocket-Node](https://github.com/theturtle32/WebSocket-Node): A WebSocket server API implementation for [Node.js](https://nodejs.org/).
* [Total.js](https://www.totaljs.com/): Web application framework for [Node.js](https://nodejs.org/en/) (Example: [WebSocket chat](https://github.com/totaljs/examples/tree/master/websocket))
* [Faye](https://www.npmjs.com/package/faye-websocket): A [WebSocket](https://developer.mozilla.org/en-US/docs/Web/API/WebSocket) (two-ways connections) and [EventSource](https://developer.mozilla.org/en-US/docs/Web/API/EventSource) (one-way connections) for [Node.js](https://nodejs.org/) Server and Client.
* [SignalR](https://dotnet.microsoft.com/en-us/apps/aspnet/signalr): SignalR will use WebSockets under the covers when it's available, and gracefully fallback to other techniques and technologies when it isn't, while your application code stays the same.
* [Caddy](https://caddyserver.com/): A web server capable of proxying arbitrary commands (stdin/stdout) as a websocket.
* [ws](https://github.com/websockets/ws): a popular WebSocket client & server library for [Node.js](https://nodejs.org/).
* [jsonrpc-bidirectional](https://github.com/bigstepinc/jsonrpc-bidirectional): Asynchronous RPC which, on a single connection, may have functions exported on the server and, and the same time, on the client (client may call server, server may also call client).
* [cowboy](https://github.com/ninenines/cowboy): Cowboy is a small, fast and modern HTTP server for Erlang/OTP with WebSocket support.
* [ZeroMQ](https://zeromq.org/): ZeroMQ is embeddable networking library that carries messages across in-process, IPC, TCP, UDP, TIPC, multicast and WebSocket.
* [WebSocket King](https://websocketking.com/): A client tool to help develop, test and work with WebSocket servers.
* [PHP WebSocket Server](https://github.com/napengam/phpWebSocketServer): Server written in PHP to handle connections via websockets wss:// or ws://and normal sockets over ssl:// ,tcp://
* [Channels](https://channels.readthedocs.io/en/stable/index.html): Django library that adds support for WebSockets (and other protocols that require long running asynchronous connections).
* [Channels](https://hexdocs.pm/phoenix/channels.html): Scalable real-time communication using WebSocket in Elixir Phoenix framework.
* [LiveView](https://github.com/phoenixframework/phoenix_live_view): Real-time interactive web experiences through WebSocket in Elixir Phoenix framework.
* [Flask-SocketIO](https://flask-socketio.readthedocs.io/en/latest/): gives Flask applications access to low latency bi-directional communications between the clients and the server.
* [Gorilla WebSocket](https://pkg.go.dev/github.com/gorilla/websocket): Gorilla WebSocket is a [Go](https://go.dev/) implementation of the WebSocket protocol.

### **File API**

**Advanced Example:** Image Editor

* **Functionality:** Allow users to upload images, edit them using canvas, and save the modified image.

**Code Snippet:** JavaScript  
const fileInput = document.getElementById('file-input');

fileInput.addEventListener('change', (e) => {

const file = e.target.files[0];

const reader = new FileReader();

reader.onload = (e) => {

const img = new Image();

img.onload = () => {

// Create a canvas and draw the image

};

img.src = e.target.result;

};

reader.readAsDataURL(file);

});

* Use code with caution.

### **Server-Sent Events**

**Advanced Example:** Real-time Stock Quotes

* **Functionality:** Display live stock quotes without constant polling.

**Code Snippet:** JavaScript  
const eventSource = new EventSource('/stock-quotes');

eventSource.onmessage = function(event) {

const data = JSON.parse(event.data);

// Update stock quotes

};

* Use code with caution.

One way websocket - only from server to client- client works exactly the same like websocket client

### **IndexedDB**

**Advanced Example:** Offline Database for E-commerce App

* **Functionality:** Store product information, user cart, and order history locally for offline access.

**Code Snippet:** JavaScript  
const dbRequest = indexedDB.open('myDatabase', 1);

dbRequest.onerror = function(event) {

console.error('Database error:', event);

};

dbRequest.onsuccess = function(event) {

const db = event.target.result;

// Create object stores and perform database operations

};

* Use code with caution.

## [**Basic pattern**](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB#basic_pattern)

The basic pattern that IndexedDB encourages is the following:

1. Open a database.
2. Create an object store in the database.
3. Start a transaction and make a request to do some database operation, like adding or retrieving data.
4. Wait for the operation to complete by listening to the right kind of DOM event.
5. Do something with the results (which can be found on the request object).

**IndexedDB databases store key-value pairs.**

**IndexedDB is built on a transactional database model. - what happen if two different instances openebd in two difference tabs**

**IndexedDB is object-oriented**

**IndexedDB does not use Structured Query Language (SQL).**

**The IndexedDB API is mostly asynchronous.**

**IndexedDB uses DOM events to notify you when results are available.**

**IndexedDB uses a lot of requests.**

**IndexedDB adheres to a same-origin policy**

**It is ODBMS.**

**Instead, you "request" that a database operation happens. You get notified by a DOM event when the operation finishes, and the type of event you get lets you know if the operation succeeded or failed.**

**Note:** These are basic examples to illustrate the concepts. Real-world applications often involve more complex logic and error handling.

**Would you like to delve deeper into a specific API or explore a more complex use case?**

## **Async vs Defer in Script Tags**

Both async and defer attributes are used in script tags to improve page loading performance by allowing the browser to continue parsing the HTML document while the script is being downloaded. However, they differ in when the script is executed.

### **Async**

* **Downloads:** The script is downloaded asynchronously, without blocking the HTML parsing process.
* **Executes:** As soon as the script is fully downloaded, it executes immediately, regardless of the parsing status of the HTML document.
* **Order:** Scripts with the async attribute are executed in the order they are downloaded, not necessarily the order they appear in the HTML.

### **Defer**

* **Downloads:** The script is downloaded asynchronously, without blocking the HTML parsing process.
* **Executes:** The script is executed after the HTML document has been fully parsed, but before the DOMContentLoaded event fires.
* **Order:** Scripts with the defer attribute are executed in the order they appear in the HTML.

### **When to use which?**

* **Async:**
  + Ideal for scripts that don't depend on the DOM structure and can execute independently.
  + Improves page loading performance by allowing the browser to render content while the script is downloading and executing.
  + Examples: analytics scripts, ad scripts.
* **Defer:**
  + Suitable for scripts that need to access the DOM and rely on the order of execution.
  + Ensures scripts execute in the correct order, preventing potential issues.
  + Examples: scripts that modify the DOM, scripts that depend on other scripts.

### **Key Differences**

| **Feature** | **Async** | **Defer** |
| --- | --- | --- |
| Downloading | Asynchronous | Asynchronous |
| Execution | Immediately after download | After HTML parsing |
| Order | Download order | Appearance order |

Export to Sheets

**In summary:**

* async prioritizes download and execution speed, but might lead to unpredictable execution order.
* defer ensures scripts execute in order after the HTML is parsed, but might slightly delay script execution.

By understanding these differences, you can optimize your website's performance and script execution based on your specific needs.

### **Intermediate Level Understanding**

**HTML Parsing Process**

When a browser receives an HTML document, it undergoes a multi-step process to render it into a visually appealing webpage:

1. **Tokenization:**
   * The browser breaks the HTML code into smaller units called tokens. These tokens represent elements, attributes, and text content.
   * For instance, <p> becomes a start tag token, </p> an end tag token, and "This is a paragraph" becomes a text token.
2. **DOM Construction:**
   * The browser builds a tree-like structure called the Document Object Model (DOM) based on the tokens.
   * Each HTML element becomes a node in the DOM tree, with child nodes for nested elements.
   * This tree represents the hierarchical structure of the HTML document.
3. **CSS Parsing:**
   * If linked stylesheets exist, the browser fetches and parses them to create the CSS Object Model (CSSOM).
   * CSSOM defines the styling rules for the DOM elements.
4. **DOM and CSSOM Merge:**
   * The browser combines the DOM and CSSOM to create a render tree.
   * This tree determines the visual representation of the page, including layout, positioning, and styling.
5. **Layout Calculation:**
   * The browser calculates the exact position and size of each element in the viewport.
   * This involves resolving dimensions, margins, padding, and other CSS properties.
6. **Painting:**
   * The browser renders the pixels onto the screen based on the layout information.
   * This is the final stage where the webpage becomes visible.

**Example:**

HTML

<!DOCTYPE html>

<html>

<head>

<title>HTML Parsing Example</title>

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

</head>

<body>

<h1>Hello, World!</h1>

<p>This is a paragraph.</p>

</body>

</html>

Use code with caution.

* The browser tokenizes the HTML into elements like <html>, <head>, <title>, <h1>, <p>, etc.
* It constructs the DOM tree with these elements as nodes.
* If the styles.css file is present, it's parsed to create the CSSOM.
* The DOM and CSSOM are merged to create the render tree.
* The browser calculates the layout, positioning the heading and paragraph accordingly.
* Finally, the browser paints the elements onto the screen.

**Intermediate Level Considerations:**

* **Asynchronous Script Loading:** Scripts can be loaded asynchronously to improve page load performance.
* **DOM Manipulation:** JavaScript can dynamically modify the DOM, affecting the rendered page.
* **CSS Preprocessors:** Tools like Sass or Less can be used to write CSS more efficiently.
* **Browser Rendering Engines:** Different browsers have different rendering engines with varying performance characteristics.

### **Advanced Level Understanding**

**HTML5 Parsing and Error Handling:**

* HTML5 introduced more flexibility in parsing, allowing for forgiving error handling.
* The parser can recover from errors and continue building the DOM.
* Understanding error recovery mechanisms is crucial for robust web applications.

**DOM Manipulation Techniques:**

* Advanced DOM manipulation techniques involve using JavaScript to create, modify, and remove elements efficiently.
* Techniques like virtual DOM and shadow DOM can optimize performance.

**CSS Layout Models:**

* Different layout models (flexbox, grid, absolute positioning) have varying complexities.
* Understanding how the browser calculates layout in each model is essential for advanced styling.

**Rendering Performance Optimization:**

* Techniques like minimizing layout thrashing, reducing reflows, and optimizing CSS can significantly improve rendering performance.
* Understanding the browser's rendering pipeline is key to optimization.

**Advanced Topics:**

* Web Components: Creating custom HTML elements with encapsulated behavior.
* Service Workers: Offline capabilities and performance improvements.
* WebAssembly: Running compiled code directly in the browser.

**By delving into these advanced concepts, you can build highly performant and interactive web applications.**

**Would you like to explore any of these topics in more detail?**

[<address>](https://www.w3schools.com/tags/tag_address.asp)

[<article>](https://www.w3schools.com/tags/tag_article.asp)

[<aside>](https://www.w3schools.com/tags/tag_aside.asp)

[<blockquote>](https://www.w3schools.com/tags/tag_blockquote.asp)

[<canvas>](https://www.w3schools.com/tags/tag_canvas.asp)

[<dd>](https://www.w3schools.com/tags/tag_dd.asp)

[<div>](https://www.w3schools.com/tags/tag_div.asp)

[<dl>](https://www.w3schools.com/tags/tag_dl.asp)

[<dt>](https://www.w3schools.com/tags/tag_dt.asp)

[<fieldset>](https://www.w3schools.com/tags/tag_fieldset.asp)

[<figcaption>](https://www.w3schools.com/tags/tag_figcaption.asp)

[<figure>](https://www.w3schools.com/tags/tag_figure.asp)

[<footer>](https://www.w3schools.com/tags/tag_footer.asp)

[<form>](https://www.w3schools.com/tags/tag_form.asp)

[<h1>-<h6>](https://www.w3schools.com/tags/tag_hn.asp)

[<header>](https://www.w3schools.com/tags/tag_header.asp)

[<hr>](https://www.w3schools.com/tags/tag_hr.asp)

[<li>](https://www.w3schools.com/tags/tag_li.asp)

[<main>](https://www.w3schools.com/tags/tag_main.asp)

[<nav>](https://www.w3schools.com/tags/tag_nav.asp)

[<noscript>](https://www.w3schools.com/tags/tag_noscript.asp)

[<ol>](https://www.w3schools.com/tags/tag_ol.asp)

[<p>](https://www.w3schools.com/tags/tag_p.asp)

[<pre>](https://www.w3schools.com/tags/tag_pre.asp)

[<section>](https://www.w3schools.com/tags/tag_section.asp)

[<table>](https://www.w3schools.com/tags/tag_table.asp)

[<tfoot>](https://www.w3schools.com/tags/tag_tfoot.asp)

[<ul>](https://www.w3schools.com/tags/tag_ul.asp)

[<video>](https://www.w3schools.com/tags/tag_video.asp)

## **Semantic HTML: A Deeper Dive**

### **What is Semantic HTML?**

Semantic HTML uses HTML tags to describe the meaning of content, rather than just its appearance. It provides a clear structure to a webpage, making it easier for both humans and machines to understand.

**Key benefits of Semantic HTML:**

* **Improved SEO:** Search engines can better understand the content and structure of the page.
* **Enhanced accessibility:** Screen readers can interpret the page more accurately.
* **Better maintainability:** The code is more readable and easier to modify.

### **Intermediate Level Examples**

**Beyond the basics:** While tags like header, nav, main, article, section, aside, footer are foundational, let's explore more advanced usage:

**figure and figcaption:**HTML  
<figure>

<img src="image.jpg" alt="Descriptive image">

<figcaption>A captivating image description.</figcaption>

</figure>

* Use code with caution.  
  This clearly defines an independent content block (the image) and its caption.

**details and summary:**HTML  
<details>

<summary>Click to expand</summary>

<p>Hidden content that expands on click.</p>

</details>

* Use code with caution.  
  Provides interactive elements for displaying additional information.

**mark:**HTML  
<p>This is some text with a <mark>highlighted</mark> word.</p>

* Use code with caution.  
  Indicates highlighted text, often used for search results or annotations.

**time:**HTML  
<time datetime="2023-11-18T13:37:00">November 18, 2023</time>

* Use code with caution.  
  Provides machine-readable information about a date and time.

### **Advanced Level Semantic HTML**

**Custom elements:** For highly specific content, define custom elements:  
HTML  
<custom-card title="Product Card">

</custom-card>

* Use code with caution.  
  With appropriate JavaScript, you can create reusable components with clear semantic meaning.

**Microdata and RDFa:** Embed structured data directly into HTML for search engines and other applications to understand:  
HTML  
<div itemscope itemtype="http://schema.org/Product">

<span itemprop="name">Product Name</span>

<span itemprop="price">$19.99</span>

</div>

* Use code with caution.

Microdata vocabularies provide the semantics or meaning of an *Item*. Web developers can design a custom vocabulary or use vocabularies available on the web, such as the widely used [schema.org](https://schema.org/) vocabulary. A collection of commonly used markup vocabularies are provided by Schema.org.

Commonly used vocabularies:

* Creative works: [CreativeWork](https://schema.org/CreativeWork), [Book](https://schema.org/Book), [Movie](https://schema.org/Movie), [MusicRecording](https://schema.org/MusicRecording), [Recipe](https://schema.org/Recipe), [TVSeries](https://schema.org/TVSeries)
* Embedded non-text objects: [AudioObject](https://schema.org/AudioObject), [ImageObject](https://schema.org/ImageObject), [VideoObject](https://schema.org/VideoObject)
* [Event](https://schema.org/Event)
* [Health and medical types](https://schema.org/docs/meddocs.html): Notes on the health and medical types under [MedicalEntity](https://schema.org/MedicalEntity)
* [Organization](https://schema.org/Organization)
* [Person](https://schema.org/Person)
* [Place](https://schema.org/Place), [LocalBusiness](https://schema.org/LocalBusiness), [Restaurant](https://schema.org/Restaurant)
* [Product](https://schema.org/Product), [Offer](https://schema.org/Offer), [AggregateOffer](https://schema.org/AggregateOffer)
* [Review](https://schema.org/Review), [AggregateRating](https://schema.org/AggregateRating)
* [Action](https://schema.org/Action)
* [Thing](https://schema.org/Thing)
* [Intangible](https://schema.org/Intangible)

Major search engine operators like Google, Microsoft, and Yahoo! rely on the [schema.org](https://schema.org/) vocabulary to improve search results. For some purposes, an ad hoc vocabulary is adequate. For others, a vocabulary will need to be designed. Where possible, authors are encouraged to re-use existing vocabularies, as this makes content re-use easier.

## [Localization](https://developer.mozilla.org/en-US/docs/Web/HTML/Microdata#localization)

In some cases, search engines covering specific regions may provide locally-specific extensions of microdata. For example, [Yandex](https://yandex.com/), a major search engine in Russia, supports microformats such as hCard (company contact information), hRecipe (food recipe), hReview (market reviews) and hProduct (product data) and provides its own format for the definition of the terms and encyclopedic articles. This extension was made to solve transliteration problems between the Cyrillic and Latin alphabets. Due to the implementation of additional marking parameters of Schema's vocabulary, the indexation of information in Russian-language web-pages became considerably more successful.

## [Global attributes](https://developer.mozilla.org/en-US/docs/Web/HTML/Microdata#global_attributes)

[itemid](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/itemid) – The unique, global identifier of an item.

[itemprop](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/itemprop) – Used to add properties to an item. Every HTML element may have an itemprop attribute specified, where an itemprop consists of a name and value pair.

[itemref](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/itemref) – Properties that are not descendants of an element with the itemscope attribute can be associated with the item using an itemref. itemref provides a list of element ids (not itemids) with additional properties elsewhere in the document.

[itemscope](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/itemscope) – The itemscope attribute (usually) works along with [itemtype](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/itemtype) to specify that the HTML contained in a block is about a particular item. The itemscope attribute creates the *Item* and defines the scope of the itemtype associated with it. The itemtype attribute is a valid URL of a vocabulary (such as [schema.org](https://schema.org/)) that describes the item and its properties context.

[itemtype](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/itemtype) – Specifies the URL of the vocabulary that will be used to define itemprop's (item properties) in the data structure. The [itemscope](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/itemscope) attribute is used to set the scope of where in the data structure the vocabulary set by itemtype will be active.

**ARIA attributes:** Enhance accessibility for screen readers by providing additional information:  
HTML  
<button aria-label="Submit form">Submit</button>

* Use code with caution.

### **Auditing a Site for Semantic Issues**

**Tools and techniques:**

* **Browser developer tools:** Inspect the HTML structure and identify misused or missing elements.
* **Lighthouse:** Provides an overall audit score and specific recommendations for semantic HTML.
* **WAVE:** Offers a visual representation of accessibility issues, including semantic problems.
* **Manual review:** Analyze content and structure to ensure appropriate tag usage.

**Common semantic issues:**

* Overuse of generic elements like div and span.
* Incorrect use of headings (e.g., multiple h1 elements).
* Missing or incorrect use of alt attributes for images.
* Lack of semantic structure for navigation, articles, and sections.

**Example audit findings:**

* A blog post uses div for sections instead of article and section.
* Headings are not used in a logical order (e.g., h3 before h2).
* Image alt attributes are missing or not descriptive.

By addressing these issues, you can significantly improve the overall quality, accessibility, and SEO of your website.

## **Semantic HTML5 Tags: A Comprehensive Guide**

HTML5 introduced a rich set of semantic elements, significantly enhancing the structure and meaning of web pages. Let's categorize them into basic, intermediate, and advanced levels based on their common usage and complexity.

### **Basic Semantic Tags**

These tags define the fundamental structure of a web page:

* **<header>**: Defines a header for a document or section.  
  + **Example:**

HTML  
<header>

<h1>My Website</h1>

<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

</header>

* Use code with caution.  
  + **Practical use:** Clearly identifies the header section of a page, often containing the site title, logo, and navigation.
* **<nav>**: Defines a section of navigation links.  
  + **Example:**

HTML  
<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

* Use code with caution.  
  + **Practical use:** Marks sections of navigation links, making it easier for users and search engines to understand the page structure.
* **<section>**: Defines a thematic grouping of content, typically with a heading.  
  + **Example:**

HTML  
<section>

<h2>About Us</h2>

<p>This is the about section.</p>

</section>

* Use code with caution.  
  + **Practical use:** Divides content into logical sections, improving readability and accessibility.
* **<article>**: Represents independent, self-contained content.  
  + **Example:**

HTML  
<article>

<h2>Blog Post</h2>

<p>This is a blog post.</p>

</article>

* Use code with caution.  
  + **Practical use:** Encapsulates self-contained content like blog posts, news articles, or forum threads.
* **<aside>**: Contains content indirectly related to the main content, like sidebars.  
  + **Example:**

HTML  
<aside>

<h2>Sidebar</h2>

<p>This is a sidebar.</p>

</aside>

* Use code with caution.  
  + **Practical use:** Separates supplementary content from the main content for better organization.
* **<footer>**: Defines a footer for a document or section.  
  + **Example:**

HTML  
<footer>

<p>&copy; Copyright 2023</p>

</footer>

* Use code with caution.  
  + **Practical use:** Clearly identifies the footer section, often containing copyright information, contact details, and links.

### **Intermediate Semantic Tags**

These tags provide more granular control over content structure:

* **<figure>**: Represents self-contained content, like illustrations, diagrams, photos, code listings, etc.  
  + **Example:**

HTML  
<figure>

<img src="image.jpg" alt="Image description">

<figcaption>A beautiful image.</figcaption>

</figure>

* Use code with caution.  
  + **Practical use:** Encapsulates media content with optional captions.
* **<details>**: Provides a disclosure element, revealing details when opened.  
  + **Example:**

HTML  
<details>

<summary>Click to expand</summary>

<p>Additional information.</p>

</details>

* Use code with caution.  
  + **Practical use:** Creates interactive elements to hide or show content.
* **<summary>**: Provides a summary for <details>.  
  + **Example:** (Used within <details>)
* **<mark>**: Highlights text.  
  + **Example:**

HTML  
<p>This is some <mark>important</mark> text.</p>

* Use code with caution.  
  + **Practical use:** Emphasizes specific text within a sentence.
* **<time>**: Represents a specific point in time.  
  + **Example:**

HTML  
<time datetime="2023-11-24T12:00:00">November 24, 2023</time>

* Use code with caution.  
  + **Practical use:** Indicates dates, times, and durations in a machine-readable format.

### **Advanced Semantic Tags**

These tags offer specialized semantics for specific content types:

* **<main>**: Represents the main content of a document.  
  + **Example:**

HTML  
<main>

<h1>Main Content</h1>

<p>This is the main content.</p>

</main>

* Use code with caution.  
  + **Practical use:** Clearly identifies the primary content of a page, helping search engines understand the page's focus.
* **<dialog>**: Represents a dialog.  
  + **Example:**

HTML  
<dialog open>

<h2>Dialog</h2>

<p>This is a dialog.</p>

</dialog>

* Use code with caution.  
  + **Practical use:** Creates custom dialog boxes for user interactions.
* **<template>**: Represents a client-side template that can be instantiated to

#### **The <template>** [**HTML**](https://developer.mozilla.org/en-US/docs/Web/HTML) **element serves as a mechanism for holding** [**HTML**](https://developer.mozilla.org/en-US/docs/Glossary/HTML) **fragments, which can either be used later via JavaScript or generated immediately into shadow DOM.**

Microdata is part of the [WHATWG](https://developer.mozilla.org/en-US/docs/Glossary/WHATWG) HTML Standard and is used to nest metadata within existing content on web pages. Search engines and web crawlers can extract and process microdata from a web page and use it to provide a richer browsing experience for users. Search engines benefit greatly from direct access to this structured data because it allows search engines to understand the information on web pages and provide more relevant results to users. Microdata uses a supporting vocabulary to describe an item and name-value pairs to assign values to its properties. Microdata is an attempt to provide a simpler way of annotating HTML elements with machine-readable tags than the similar approaches of using RDFa and classic microformats.

At a high level, microdata consists of a group of name-value pairs. The groups are called items, and each name-value pair is a property. Items and properties are represented by regular elements.

* To create an item, the itemscope attribute is used.
* To add a property to an item, the itemprop attribute is used on one of the item's descendants.

### **Understanding HTML Outlines**

An HTML outline is a hierarchical representation of the structure of a document. It's essentially a roadmap for both humans and machines to understand the content's organization. It's crucial for accessibility, SEO, and overall document clarity.

### **Outlines in HTML4 vs. HTML5**

**HTML4:**

* Primarily relied on heading elements (h1 to h6) to define the document structure.
* Lacked dedicated elements for sections, articles, aside, nav, etc.
* Often used div and span elements with classes for semantic purposes, which could lead to inconsistencies.

**HTML5:**

* Introduced a rich set of semantic elements to accurately represent document structure.
* Provides elements like <header>, <nav>, <section>, <article>, <aside>, <footer> for clear delineation of content.
* Enhances accessibility and SEO by providing explicit semantic meaning to different parts of the document.

### **Intermediate Example: Basic Structure**

HTML

<!DOCTYPE html>

<html>

<head>

<title>My Web Page</title>

</head>

<body>

<header>

<h1>My Website</h1>

<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

</header>

<main>

<article>

<h2>Article Title</h2>

<p>Content of the article.</p>

</article>

<aside>

<h3>Sidebar</h3>

<p>Additional content.</p>

</aside>

</main>

<footer>

<p>&copy; Copyright 2024</p>

</footer>

</body>

</html>

Use code with caution.

### **Advanced Example: Complex Structure**

HTML

<!DOCTYPE html>

<html>

<head>

<title>Complex Structure Example</title>

</head>

<body>

<header>

</header>

<main>

<article>

<h2>Main Article</h2>

<section>

<h3>Section 1</h3>

<p>Content of section 1.</p>

</section>

<section>

<h3>Section 2</h3>

<p>Content of section 2.</p>

</section>

</article>

<aside>

<nav>

<h3>Site Navigation</h3>

</nav>

<section>

<h3>Related Content</h3>

</section>

</aside>

</main>

<footer>

</footer>

</body>

</html>

Use code with caution.

## **Text-Level Semantic Elements in HTML5**

Text-level semantic elements are used to define specific text styles and meanings within a document. HTML5 offers a rich set of these elements to improve content structure, accessibility, and search engine optimization (SEO).

### **Core Text-Level Elements**

* **<a> (Anchor):** Defines a hyperlink.
  + Example: <a href="https://example.com">Link to Example</a>
  + Purpose: Creates clickable links to other documents or locations within the same document.
* **<em> (Emphasis):** Emphasizes text without conveying specific meaning.
  + Example: <em>This text is emphasized.</em>
  + Purpose: Highlights text for stylistic reasons, often rendered in italics.
* **<strong> (Strong):** Indicates strong importance.
  + Example: <strong>This text is important.</strong>
  + Purpose: Highlights text to convey strong importance, often rendered in bold.
* **<small>:** Renders text in a smaller font size.
  + Example: <small>This text is small.</small>
  + Purpose: Indicates smaller-scale text, often used for disclaimers or copyright information.
* **<del> (Deleted):** Indicates text that has been deleted.
  + Example: <del>This text is deleted.</del>
  + Purpose: Marks text as deleted, often used for version control or strikethrough effect.
* **<ins> (Inserted):** Indicates text that has been inserted.
  + Example: <ins>This text is inserted.</ins>
  + Purpose: Marks text as inserted, often used for version control or underline effect.
* **<code>:** Renders computer code.
  + Example: <code><p>Hello, world!</p></code>
  + Purpose: Displays code or other computer output in a monospace font.
* **<var>:** Represents a variable.
  + Example: <var>x</var> = 10
  + Purpose: Indicates a variable or mathematical expression.
* **<samp>:** Represents sample output from a program.
  + Example: <samp>User input: hello</samp>
  + Purpose: Displays sample output from a program, often in a monospace font.
* **<kbd>:** Represents user input.
  + Example: <kbd>Press Ctrl+S to save.</kbd>
  + Purpose: Indicates text that should be entered by the user, often in a monospace font.
* **<sub> (Subscript):** Renders text as subscript.
  + Example: H<sub>2</sub>O
  + Purpose: Creates subscript text, commonly used in chemical formulas.
* **<sup> (Superscript):** Renders text as superscript.
  + Example: x<sup>2</sup>
  + Purpose: Creates superscript text, commonly used for exponents or footnotes.
* **<abbr> (Abbreviation):** Defines an abbreviation or acronym.
  + Example: <abbr title="World Wide Web">WWW</abbr>
  + Purpose: Provides an explanation for an abbreviation, displayed as a tooltip on hover.
* **<time>:** Represents a specific time or date.
  + Example: <time datetime="2024-07-21">July 21, 2024</time>
  + Purpose: Indicates a specific time or date, with machine-readable format for parsing.
* **<cite>:** Indicates a citation.
  + Example: <cite>The Elements of Style</cite>
  + Purpose: Marks the title of a work, often rendered in italics.
* **<q> (Quotation):** Represents a short quotation.
  + Example: <q>She said, "Hello."</q>
  + Purpose: Indicates a short quotation, often rendered with quotation marks.
* **<dfn> (Definition):** Defines a term.
  + Example: <dfn>HTML</dfn> is a markup language.
  + Purpose: Introduces a term and its definition.

### **Advanced Text-Level Elements**

* **<mark>:** Highlights text, often used for search results.
  + Example: <mark>This text is highlighted.</mark>
  + Purpose: Indicates highlighted text, typically with a background color.
* **<ruby>:** Represents ruby annotations (used in East Asian languages).
  + Example:

HTML  
<ruby>漢<rp>(</rp><rt>かん</rt><rp>)</rp></ruby>

* Use code with caution.
  + Purpose: Provides pronunciation or other information for characters.
* **<rt> (Ruby Text):** Contains the ruby text.
* **<rp> (Ruby Parenthesis):** Optional element to surround the ruby text.
* **<bdi> (Bidirectional Isolation):** Isolates a part of text that has different directionality.
  + Example: <bdi>right-to-left text</bdi>
  + Purpose: Prevents text directionality from affecting surrounding text.
* **<bdo> (Bidirectional Override):** Overrides the directionality of text.
  + Example: <bdo dir="rtl">This text is right-to-left.</bdo>
  + Purpose: Forces text directionality, regardless of surrounding text.
* **<span>:** Generic inline element for grouping inline elements.
  + Example: <span class="highlight">Important text</span>
  + Purpose: Provides a container for styling or scripting purposes.
* **<br>:** Inserts a single line break.
  + Example: <br>
  + Purpose: Forces a line break without starting a new paragraph.
* **<wbr> (Word Break Opportunity):** Suggests a possible line break within a word.
  + Example: <wbr>verylongword</wbr>
  + Purpose: Prevents overly long words from breaking lines in an undesirable way.

By understanding and effectively using these text-level semantic elements, you can create more accessible, structured, and SEO-friendly web content.

## **Semantic Forms in HTML5**

While HTML5 doesn't introduce new specific elements for forms beyond the traditional form, input, select, textarea, etc., it emphasizes the importance of **semantic structure** within forms. This means using elements effectively to convey the meaning and purpose of different form sections.

### **Key Semantic Considerations for Forms**

1. **Form Structure:**
   * **<fieldset>:** Groups related form elements together.
   * **<legend>:** Provides a caption for the fieldset, describing its purpose.
   * **<label>:** Associates a label with a form element for accessibility and usability.
2. **Input Types:**
   * Utilize the various type attributes of the input element to accurately represent the input type (text, email, password, number, date, etc.).
   * Consider using required and placeholder attributes for clarity.
3. **Form Validation:**
   * Employ HTML5 input types and patterns for basic client-side validation.
   * Implement custom JavaScript validation for more complex rules.
4. **Accessibility:**
   * Use descriptive labels for all form elements.
   * Provide clear error messages.
   * Ensure proper keyboard navigation.

### **Example**

HTML

<form>

<fieldset>

<legend>User Information</legend>

<label for="name">Name:</label>

<input type="text" id="name" name="name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

</fieldset>

<fieldset>

<legend>Password</legend>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

</fieldset>

<button type="submit">Submit</button>

</form>

Use code with caution.

### **Additional Tips**

* Use clear and concise labels.
* Group related form elements together using fieldsets.
* Provide helpful placeholder text.
* Consider using input masks for specific formats (e.g., phone numbers, dates).
* Test your forms thoroughly on different browsers and devices.

### **Beyond Basic Structure**

While HTML5 itself doesn't provide revolutionary new form elements, it encourages best practices and semantic markup. By following these guidelines, you can create forms that are not only visually appealing but also accessible, user-friendly, and maintainable.

## **New HTML5 Input Types with Validations**

**HTML5 introduced several new input types to improve user experience and simplify form validation. Let's explore these new types, their validations, and related elements:**

### **New Input Types**

* **email: Validates email addresses.**
* **url: Validates URLs.**
* **tel: For phone numbers (validation is less strict).**
* **number: For numeric input, with optional min, max, and step attributes.**
* **range: For a slider-like input, with min, max, and step attributes.**
* **date: For selecting a date, with min and max attributes.**
* **month: For selecting a month, with min and max attributes.**
* **week: For selecting a week, with min and max attributes.**
* **time: For selecting a time, with min, max, and step attributes.**
* **datetime-local: For selecting a date and time.**
* **color: For selecting a color.**
* **search: For search fields.**

### **Exposing Validation Messages**

* **required attribute: Makes a field mandatory.**
* **pattern attribute: Defines a regular expression for validation.**
* **:valid and :invalid pseudo-classes: Style elements based on validation state.**

**Example:**

**HTML**

**<form>**

**<label for="email">Email:</label>**

**<input type="email" id="email" name="email" required>**

**<span class="error" aria-live="polite"></span>**

**</form>**

**Use code with caution.**

**CSS**

**input:invalid + .error {**

**content: "Invalid email address";**

**color: red;**

**}**

**Use code with caution.**

### **Datalist**

* **<datalist>: Provides a suggestion list for <input> elements.**
* **<option>: Defines options within the datalist.**

**Example:**

**HTML**

**<input list="browsers">**

**<datalist id="browsers">**

**<option value="Chrome">**

**<option value="Firefox">**

**<option value="Opera">**

**<option value="Safari">**

**</datalist>**

**Use code with caution.**

### **Pseudo-Classes**

* **:valid: Applies to valid form elements.**
* **:invalid: Applies to invalid form elements.**
* **:required: Applies to required form elements.**
* **:optional: Applies to optional form elements.**
* **:disabled: matches with disabled**

**Example:**

**CSS**

**input:invalid {**

**border-color: red;**

**}**

**Use code with caution.**

### **Additional Considerations**

* **Browser Compatibility: While HTML5 input types are widely supported, there might be differences in behavior and styling across browsers.**
* **Accessibility: Ensure proper labeling and error messages for accessibility.**
* **Custom Validation: For complex validation scenarios, use JavaScript.**
* **Fallback: Provide fallback options for older browsers that don't support new input types.**

**By effectively using these new input types and validation features, you can create more user-friendly and efficient forms.**