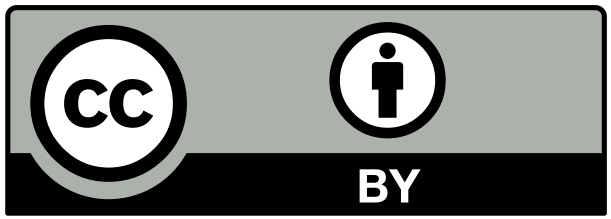
HTML

Péter Jeszenszky

Faculty of Informatics, University of Debrecen

October 27, 2025

Terms of Use



Markup Languages (1)

- Markup languages are computer languages for annotating text.
- They allow the association of metadata with parts of text in a clear distinguishable way.

Markup Languages (2)

- The term “markup” refers to the syntactical constructs available to associate metadata with parts of text.
  - They can be used to indicate the meaning of a piece of text, or to control how processing applications should render it.
- The term “document” refers to text intermingled with markup.

What is HTML? (1)

- Originally, HTML stood for HyperText Markup Language.
- However, the current HTML standard no longer treats “HTML” as an abbreviation; it uses it as a standalone term without expansion.

What is HTML (2)

- "HTML is the World Wide Web's core markup language."
- "[...] a semantic-level markup language and associated semantic-level scripting APIs for authoring accessible pages on the Web ranging from static documents to dynamic applications."
- Source: HTML Living Standard

What is HTML? (3)

- HTML is a markup language whose purpose is to publish content on the Web.
- It can be used to create both static web pages and dynamic web applications.
- The term “hypertext” refers to the capability of representing navigable relationships between documents, also called hyperlinks.

The Birth of HTML

- Tim Berners-Lee (TBL) invented and created HTML.
- The first public website: http://info.cern.ch/ (launched on: August 6, 1991)
  - See: Restoring the first website
- The first publicly available technical documentation about HTML: https://info.cern.ch/hypertext/WWW/MarkUp/Tags.html
- The early versions of HTML were all based on SGML.

Major Versions of HTML (1)

- HTML 4.01 (superseded)
- XHTML 1.0 (superseded)
- XHTML 1.1 (superseded)
- HTML5 (current version)

Major Versions of HTML (2)

Version statistics:

- Usage statistics and market share of HTML for websites (W3Techs)
  HTML5 is used by 97.8% of all the websites who use HTML.

HTML 4.01 (1)

HTML 4.01 Specification (W3C Recommendation, 24 December 1999; superseded: 27 March 2018)

- The last SGML-based HTML version
- Media type: `text/html`

HTML 4.01 (2)

Document type declarations:

- **Strict**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

- **Transitional**:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

- **Frameset**:

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Frameset//EN"
  "http://www.w3.org/TR/html4/frameset.dtd">
```

XHTML

- HTML defined as an XML application imposes stricter constraints on documents, thus, they can be processed more easily.
- This is particularly important for devices that have limited capabilities compared to desktop computers (e.g., mobile devices).
- XHTML and its modularization enable the combination of XHTML with other XML applications.
  - For example, embedding MathML and SVG in XHTML documents – however, the resulting documents are no longer considered to be XHTML documents.

XHTML 1.0 (1)

XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) – A Reformulation of HTML 4 in XML 1.0 (W3C Recommendation, 26 January 2000; superseded: 27 March 2018)

- A reformulation of HTML 4 as an XML application
- Media type: `application/xhtml+xml`

XHTML 1.0 (2)

Document type declarations:

- **Strict**:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- **Transitional**:

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- **Frameset**:

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Frameset//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

XHTML 1.0 (3)

Web pages written in XHTML 1.0:

- IANA protocol registries, e.g.:
  - Hypertext Transfer Protocol (HTTP) Status Code Registry
  - Media Types

XHTML 1.1 (1)

XHTML Modularization 1.1 – Second Edition (W3C Recommendation, 29 July 2010; superseded: 27 March 2018)

- Modularization provides a means for subsetting and extending XHTML.
- Can be implemented using either DTD or XML Schema.
- Provides a set of standard modules.
  - Examples: *Frames* (`frame`, `frameset`, `noframes` elements), *Hypertext* (`a` element), *Text* (`div`, `h1`, `p`, ... elements), ...
- Combining multiple modules enables the creation of so-called hybrid document types.

XHTML 1.1 (2)

XHTML Basic 1.1 – Second Edition (W3C Recommendation, 23 November 2010; superseded: 27 March 2018)

- A subset of XHTML suitable to be used on a wide range of devices (e.g., mobile phones, PDAs, e-book readers, set top boxes).
- The document type definition is implemented using XHTML modules as defined in the *XHTML Modularization* recommendation.
- Document type declaration:

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML Basic 1.1//EN"
  "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

- Media type: `application/xhtml+xml`

XHTML 2.0

XHTML 2.0 (W3C Working Group Note, 16 December 2010)

- A new language that was not compatible with the earlier HTML and XHTML languages.
- The W3C decided to not continue the development of XHTML 2.0.
- HTML5 has taken over its originally intended role.
- See: Frequently Asked Questions (FAQ) about the future of XHTML (W3C)

HTML5: History (1)

See: HTML Living Standard – Introduction – History

HTML5: History (2)

- Originally, the HTML5 specification was developed by the WHATWG.
- The W3C joined to the development of HTML5 in 2007.
- See:
  - Ian Hickson. The WHATWG Blog – W3C restarts HTML effort. 7 March 2007.
  - W3C Relaunches HTML Activity. 7 March 2007.

HTML5: History (3)

- Between July 2012 and June 2019, the WHATWG and the W3C had been working on separate specifications that followed a different development model.
- The W3C started working on the next version (HTML 5.1) after the publication of the HTML 5.0 specification as a Recommendation.
- The WHATWG specification will never be completed, it is under continuous development ("living standard").

HTML5: History (4)

- Until June 2019, within the W3C the Web Platform Working Group developed the specifications related to the HTML language.
- The W3C's specifications was based on the WHATWG's specification.
  - The W3C had published certain parts in separate documents.

HTML5: History (5)

- On May 28, 2019, the two organizations signed an agreement to collaborate on the development of a single version of the HTML and DOM specifications.
  - See: Jeff Jaffe. W3C and WHATWG to work together to advance the open Web platform. 28 May 2019.
- Collaboration agreement:
  - HTML and DOM shall be developed principally in the WHATWG.
  - W3C intends to approve and release the WHATWG specifications as W3C recommendations.
  - See: Memorandum of Understanding Between W3C and WHATWG. May 28, 2019.
- In the following, within the W3C the HTML Working Group is responsible for the development of HTML.

HTML5 Standard

- **WHATWG**:
  - HTML Living Standard
  - HTML: The Living Standard – Edition for Web Developers
    - Removes information that only browser vendors need to know.
- **W3C**: Currently, the URI https://www.w3.org/TR/html is redirected to the WHATWG specification.

HTML Document Structure

- Like XML, HTML originates from SGML.
- Thus, both XML and HTML documents are made up of elements.
- Some of HTML's constructs may seem strange or obsolete; for example, the DOCTYPE declaration.
  - These legacy features originate from SGML, on which early versions of HTML were based, and are retained primarily for backward compatibility.

Elements (1)

- HTML documents are made up of elements delimited by **start tags** and **end tags**.
  - The part of the document between a start tag and its matching end tag is called the **content** of the element.
    - The content may include both text and other elements, i.e., elements can be nested.
  - Elements can have name-value pairs carrying metadata referred to as **attributes**.
    - The attributes are specified in the start tags of elements.

Elements (2)

Examples:

- ```
  <h2>Table of contents</h2>
  ```
- ```
  <a href="https://html.spec.whatwg.org/" target="_blank">HTML
    Standard</a>
  ```
- ```
  <ul class="links">
    <li><a href="https://whatwg.org/">WHATWG</a></li>
    <li><a href="https://www.w3.org/">W3C</a></li>
    <li><a href="https://developer.mozilla.org/">MDN Web Docs</a></li>
  </ul>
  ```
  ```

HTML Documents are Trees (1)

- According to the HTML Standard, an HTML document is a tree structure, in which nodes represent an element, text, or some other HTML construct (e.g., a comment).
- Such a tree representation of an HTML document is called a **DOM tree**.
- When a browser loads an HTML document, it constructs a DOM tree in memory.
- The DOM tree can be viewed in the browser's DevTools.

HTML Documents are Trees (2)

- The nodes of the DOM tree are objects with an API through which the document can be manipulated.
  - This serves as a basis for creating dynamic web pages.
- The HTML Standard provides two concrete syntaxes to serialize and persistently store DOM trees, i.e., the HTML and the XML syntaxes.

Characters

- HTML documents are made up of Unicode characters.
- Documents must be serialized using the UTF-8 character encoding.

The Building Blocks of HTML Documents

- The following building blocks are available, each of which is represented by a node in the DOM trees:
  - Text
  - CDATA sections
  - Elements
  - Document type declaration (DOCTYPE)
  - Comments
- Not all of them are available in both concrete syntaxes.
- Concrete syntaxes can differ in how they represent them.

Text (1)

- Text can occur in element content, in comments, and in attribute values.
- Text that occurs in element content is represented by a text node in DOM trees.
  - When the DOM tree is constructed, each continuous run of text inside an element is represented by a single text node.

Text (2)

- Note that the < and the & characters are special characters.
- As a rule of thumb, they must be escaped in text.
  - They can be used literally in comments and CDATA sections.
- Example:

```
<!-- Fish emoticon, i.e., <>< -->
<code>&lt;>&lt;</code>
```

Text (3)

- In both concrete syntaxes, Unicode characters can be specified in text using decimal or hexadecimal **numeric character references**, such as `&#8364;` and `&#x20AC;` that both refer to the Euro Sign (U+20AC).
- In the HTML syntax, many Unicode characters can be specified using **named character references** of the form `&name;`, such as `&euro;`.
  - The list of character reference names: https://html.spec.whatwg.org/multipage/named-characters.html
  - In XML, these references are called **entity references**, and only five of them are available in the XML syntax, i.e., `&`, `&apos;`, `>`, `<`, and `&quot;`.

Text (4)

- All forms of character references are expanded before the DOM tree is constructed; thus, they are not present in the DOM tree.

CDATA Sections (1)

- CDATA sections can be used in text that occurs in element content to specify unescaped text.
  - Within a CDATA section, the < and the & character lose their special meaning; thus, they can occur literally.
- CDATA sections are available in the XML syntax only, without limitations.
- In the HTML syntax, their use is very limited.
  - CDATA sections are available in the HTML syntax only in the `math` and `svg` elements.
- Each CDATA section is represented by a specific node in the DOM tree.

CDATA Sections (2)

- CDATA sections are helpful in embedding computer program code in which the < and & characters are likely to occur frequently.
- Example for a CDATA section:

```
<pre><![CDATA[unsigned getbits(unsigned x, int p, int n) {
  return (x >> (p+1-n)) & ~(~0 << n);]]></pre>
}
```

Elements (1)

- Elements are the primary building blocks of HTML documents, whose nesting establishes the structure of the document.
- An element is a logical part of a document with a meaning that has the following:
  - A name clearly indicating the purpose and the meaning of its content
  - Zero or more attributes (optional)
  - Content (optional)
- Each element is represented by an element node in the DOM tree.

Elements (2)

- HTML defines the available elements, specifying the attributes they accept and the content allowed inside each of them.

Elements (3)

- Elements may have content.
  - Elements that can have content are called **non-void elements**.
    - Most of the HTML elements (e.g., `a`, `div`, `p`, or `table`) are non-void.
  - Elements that can't have content are called **void elements**.
    - Void elements include, for example, the `base` element and the `img` element.
- The nodes representing the content of an element are the children of the element node in the DOM tree.

Elements (4)

- Each element has a **content model**: a description of the element's expected contents.
- An HTML element must have contents that match the requirements described in the element's content model.

Elements (5)

- Elements are delimited by tags in the concrete syntaxes.
- The two major types of tags are **start tags** and **end tags**.
  - A third type of tag, called a **self-closing tag** of the form `<name/>`, is available only in the XML syntax.
- Elements should not be confused with tags; the latter are markup constructs that delimit the elements in the concrete syntaxes.
  - Sometimes the terms element and tag are used as synonyms; however, this is inaccurate.

Attributes (1)

- Attributes for elements can be specified in the start tags, or, in the XML syntax, in the self-closing tags.
- Their general form is either `name="value"` or `name='value'`, where the delimiter surrounding the value can't occur directly inside the value.
- In general, browsers do not display attributes directly.
  - An exception is the `title` global attribute, whose value is shown as a tooltip.
- Attributes are not present in the DOM tree as nodes; instead, they are stored as properties of the element nodes.

Attributes (2)

- In the concrete syntaxes, the attributes can be specified in any order.
- Each attribute can occur at most once on an element.
- The HTML syntax provides some extra features for attributes, such as the boolean attributes.

Attributes (3)

Attributes can be used for various purposes:

- They can carry metadata:

```
<blockquote cite="https://whatwg.org/faq">
  <p>The Web Hypertext Application Technology Working Group
  (WHATWG) is a community of people interested in evolving
  the web through standards and tests.</p>
</blockquote>
```

- They can refine the meaning of the elements:

```
<input type="text">
<input type="file">
```

- They can control the appearance or behavior of the elements:

```
<textarea name="review" readonly>Great for the price!</textarea>
```

Whitespace in Elements (1)

- When an element's content includes elements, whitespace characters are allowed to be used before, after, and between child elements.
- Such whitespace characters are often used to show the nesting of elements.
- These whitespaces are also present in the DOM tree; however, they are typically not important, thus, they are not shown in DevTools.
- In practice, web pages, mainly automatically generated pages, often lack whitespace characters, showing the nesting of elements to minimize page size.

Whitespace in Elements (2)

Examples for “single-line” web pages:

- https://lodash.com/
- https://alvarotrigo.com/fullPage/

DOCTYPE

- The DOCTYPE (i.e., `<!DOCTYPE html>`) may appear at the very beginning of HTML documents before the root element.
- It does not serve as a version identifier!
- It is considered an old relic inherited from SGML.
- Its only purpose is to switch web browsers into the proper rendering mode.
- The DOCTYPE is represented by a specific node in DOM trees.

Comments (1)

- A comment represents text to be ignored by user agents.
- In both concrete syntaxes, comments are specified between the `<!--` and the `-->` delimiters.
- Comments can serve various purposes:
  - **Documentation**: They can add an explanation to the HTML code that helps developers understand it.
  - **Reminders**: They can indicate that some part of the code is missing or needs fixing (TODO comments).
  - **Commenting out code**: Wrapping parts of the code in comments will instruct user agents to ignore them.
- Each comment is represented by a comment node in the DOM trees.

Comments (2)

Examples of comments:

- ```
  <!-- Intentionally left empty -->
  ```

- ```
  <!-- Google Tag Manager -->
  <script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
  new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
  j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src=
  'https://www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j
  })(window,document,'script','dataLayer','GTM-XXXXXX');</script>
  <!-- End Google Tag Manager -->
  ```

- ```
  <!-- TODO: add navigation links -->
  ```

- ```
  <!--
  <script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
  -->
  ```

Comments (3)

Conditional comments:

The content of this slide is communicated only orally in the lecture.

- See: Peter-Paul Koch. Conditional comments.

Comments (4)

Conditional comments:

- ```
  <!--[if IE]>
  IE-specific HTML code
  <![endif]-->
  ```

- ```
  <!--[if IE 6]>
  IE 6-specific HTML code
  <![endif]-->
  ```

- ```
  <!--[if !IE]> -->
  HTML code to be ignored by any version of IE
  <!-- <![endif]-->
  ```
  ```

Comments (5)

Conditional comments:

- Although Internet Explorer is now a discontinued product that is no longer supported, some websites still use conditional comments.
- Examples of websites using conditional comments:
  - https://www.ietf.org/
  - https://kernel.org/
  - https://www.mozilla.org/
  - https://www.python.org/

Element Semantics (1)

- Elements, attributes, and attribute values in HTML are defined to have certain meanings (semantics).
  - For example, the `ol` element represents an ordered list, and the `lang` attribute represents the language of the content.
- Authors must not use elements, attributes, or attribute values for purposes other than their appropriate intended semantic purpose.

Element Semantics (2)

- The majority of presentational features from previous versions of HTML are no longer allowed.
- The problems of presentational markup:
  - The use of presentational elements leads to poorer accessibility.
  - Higher cost of maintenance.
  - Larger document sizes.
- The only remaining presentational markup features in HTML are the `style` attribute and the `style` element.

Element Semantics (3)

The following elements that were previously presentational have been redefined to be media-independent:

Element	HTML 4.01, XHTML 1.0	HTML5
`b`	Bold font	Keywords
`i`	Italic font	Alternate voice
`hr`	Horizontal rule	Thematic break
`small`	Smaller font	Side comment
`s`	Strike-through	Inaccurate text
`u`	Underline	Unarticulated annotation (e.g., misspelt text)

Semantic HTML (1)

Example:

- ```
  <h1>This is a top-level heading</h1>
  ```
- ```
  <span style="display: block; font-size: 2em;
      font-weight: bold; margin: 0.67em 0;">
    This is not a top-level heading, although it looks
    like one
  </span>
  ```

Source: Semantics in HTML (MDN)

Semantic HTML (2)

Benefits include:

- Good for Search Engine Optimization (SEO)
- Improves maintainability
- Improves web accessibility

See: HTML: A good basis for accessibility (MDN)

Semantic HTML (3)

Related concept: **web accessibility**

- Web accessibility means that websites, tools, and technologies are designed and developed so that people with disabilities can use them.
- See: https://www.w3.org/mission/accessibility/

Categories of Elements (1)

Each element in HTML falls into zero or more of the following categories:

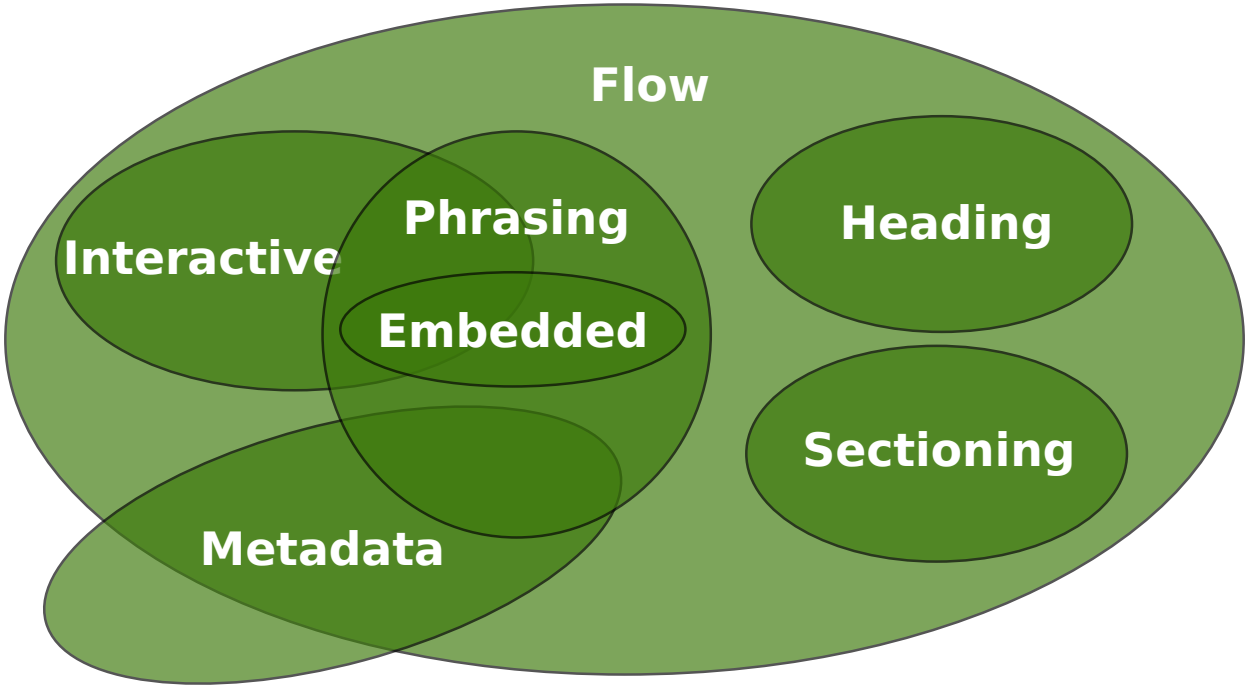


Figure 1: Source: https://html.spec.whatwg.org/images/content-venn.svg

Categories of Elements (2)

Foreign elements: Elements from the MathML namespace and the SVG namespace.

New Elements of HTML5 (1)

Elements introduced in HTML5 for better structuring of the documents:

- `article`
- `aside`
- `figure`
- `footer`
- `header`
- `hgroup`
- `main`
- `nav`
- `section`

New Elements of HTML5 (2)

Other new elements introduced in HTML5:

- `audio`
- `canvas`
- `data`
- `dialog`
- `math`
- `meter`
- `picture`
- `progress`
- `summary/details`
- `time`
- `video`
- …

HTML Elements Reference

All HTML elements:

- HTML Standard – Index – Elements
- MDN Web Docs – HTML Elements Reference

Global Attributes

Global attributes are the attributes that can be specified on all HTML elements:

- `class`
- `dir`
- `id`
- `lang`
- `style`
- `title`
- `xml:lang` (should be used only in the XML syntax)
- Custom data attributes
- ...

See: Global attributes (MDN)

Global Attributes: the `id` Attribute (1)

The content of this slide is communicated only orally in the lecture.

Global Attributes: the id Attribute (2)

- Example of use:

```
<nav id="toc">
  <h1>Table of Contents</h1>
</nav>
```

  - Using the id as a fragment identifier:

```
<a href="#toc">Jump to Table of Contents</a>
```

  - Using the id as a selector in CSS:

```
#toc {
  background-color: lavender;
}
```

Global Attributes: the `class` Attribute

- Authors can use the `class` attribute to extend elements, effectively creating their own elements, while using the most applicable existing "real" HTML element.
- When specified on HTML elements, the `class` attribute must have a value that is a set of space-separated tokens representing the various classes that the element belongs to.
- Examples of use:

```
<p class="author">...</p>
<p class="note">...</p>
<p class="warning">...</p>
<p class="note important">...</p>
```

Global Attributes: Custom Data Attributes (1)

- A **custom data attribute** is an attribute whose name starts with the string `data-` and has at least one character after the hyphen.
- Intended to store custom data, state, annotations, and similar, private to the page or application, for which there are no more appropriate attributes or elements.
- Every HTML element may have any number of custom data attributes specified, with any value.

Global Attributes: Custom Data Attributes (2)

Example of use:

```
<form>
  <p>
    <label for="house">House:</label>
    <select name="house" id="house" required>
      <option value="">--Please choose an option--</option>
      <option value="gryffindor"
        data-color-primary="red"
        data-color-secondary="gold">Gryffindor</option>
      <!-- ... --->
    </select>
  </p>
</form>
```

Global Attributes: Custom Data Attributes (3)

Example of use:

```
document.querySelectorAll('a').forEach(function (element) {
  element.addEventListener('mouseenter', function (event) {
    const a = event.currentTarget;
    const timer = setTimeout(function () {
      window.location.href = a.href;
    }, 3000);
    // Storing the timer in the data-timer attribute:
    a.dataset.timer = timer;
  });
  element.addEventListener('mouseleave', function (event) {
    // Getting the timer from the data-timer attribute:
    const timer = event.currentTarget.dataset.timer;
    clearTimeout(timer);
  });
});
```

Document Metadata (1)

- The `head` element contains metadata about the document.
- Each document needs a single `head` element.
- The following elements are available in the `head` element:
  - `base`
  - `link`
  - `meta`
  - `noscript`
  - `script`
  - `style`
  - `title`
- The content of the `head` element is not shown by the web browsers except the `title`.

Document Metadata (2)

Further information:

- Josh Buchea. HEAD: A simple guide to HTML <head> elements.

Document Metadata: the base Element

- The base element is a void element that specifies the base URI for resolving relative references.
- Each document must contain at most one base element.
- The base element is rarely used on large public websites.
- See: <base>: The Document Base URL element (MDN)
- Example of use:

```
<base href="https://www.eg.com/">
```

Document Metadata: the `link` Element (1)

The content of this slide is communicated only orally in the lecture.

- See: `<link>`: The External Resource Link element (MDN)

Document Metadata: the `link` Element (2)

Examples of use:

- ```
  <link rel="stylesheet" href="screen.css">
  ```
- ```
  <link rel="icon" href="favicon.png">
  ```
- ```
  <link rel="license"
    href="https://creativecommons.org/licenses/by/4.0/">
  ```
  ```

Document Metadata: the `meta` Element (1)

The content of this slide is communicated only orally in the lecture.

- See:
  - `<meta>`: The metadata element (MDN)
  - Meta Tags and Attributes that Google Supports (Google Search Central)

Document Metadata: the meta Element (2)

Examples of use:

- `<meta charset="UTF-8">`
- `<meta name="author" content="Tim Berners-Lee">`
- `<meta http-equiv="Refresh" content="300">`

Document Metadata: the `noscript` Element (1)

The content of this slide is communicated only orally in the lecture.

- See: `<noscript>`: The Noscript element (MDN)

Document Metadata: the `noscript` Element (2)

Example of use:

```
<noscript>
  <link rel="stylesheet" href="noscript.css">
</noscript>
```

Document Metadata: the `script` Element (1)

- The `script` element embeds or references executable (i.e., JavaScript) code.
- Each document can contain zero or more `script` elements in the `head` element.
- See: `<script>`: The Script element (MDN)

Document Metadata: the `script` Element (2)

Examples of use:

- ```
  <script>
    const startTime = performance.now();
    document.addEventListener("DOMContentLoaded", () => {
      const loadTime = (performance.now() - startTime).toFixed(2);
      console.log(`Page loaded in ${loadTime} ms`);
    });
  </script>
  ```

- ```
  <script src="js/metadata.min.js"></script>
  ```

Document Metadata: the `style` Element (1)

The content of this slide is communicated only orally in the lecture.

- See: `<style>`: The Style Information element (MDN)

Document Metadata: the `style` Element (2)

Example of use:

```
<style>
  ol, ul {
    padding: 0;
    margin: 0 0 1em 2em;
  }
  li {
    margin: 0 0 0.25em 0;
  }
</style>
```

Document Metadata: the `title` Element

- The `title` element provides the title of the document.
- Each document must contain at most one `title` element.
- See: `<title>`: The Document Title element (MDN)
- Example of use:

```
<title>HTML Standard, Edition for Web Developers</title>
```

Document Metadata: Practical Use

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>The Mysteries of HTML</title>
  <meta name="description"
    content="Everything you'd like to know about HTML, but are afraid to ask.">
  <meta name="keywords" content="faq,html,markup">
  <meta name="robots" content="index,follow">
  <meta name="googlebot" content="notranslate">
  <link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/@picocss/pico@2/css/pico.min.css">
  <link rel="icon" type="image/svg+xml" href="favicon.svg">
  <script
    src="https://cdn.jsdelivr.net/npm/jquery@3.7.1/dist/jquery.min.js"
    defer></script>
</head>
```

No Schema for HTML

- DTDs and XML schemas cannot express all the syntax and structural requirements of HTML.
- See, for example, the custom `data-*` attributes.
  - See: Embedding custom non-visible data with the data-* attributes

Concrete Syntaxes (1)

- HTML is an abstract language in which documents are represented as trees.
- The standard provides two concrete syntaxes to serialize documents, namely, the HTML and the XML syntaxes.

Concrete Syntaxes (2)

- **HTML syntax**:
  - While it bears a close resemblance to SGML and XML, it is a separate language with its own parsing rules.
  - It is compatible with most legacy web browsers.
  - File extension: `.html`, `.htm`
  - Media type: `text/html`
- **XML syntax**:
  - It is a syntax based on the XML 1.0 and the *Namespaces in XML 1.0* standards.
  - It does not define any syntax-level requirements beyond those defined for XML.
  - It is also called the XHTML syntax.
  - File extension: `.xhtml`, `.xht`
  - Media type: `application/xhtml+xml`

HTML Syntax: Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Example Page</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="screen.css">
  </head>
  <body>
    <p>Visit W3C's website by clicking on the logo:</p>
    <a href="https://www.w3.org/">
      <img src="w3c.png" width="120" height="120"
        alt="W3C logo">
    </a>
  </body>
</html>
```

HTML Syntax: DOCTYPE

- In the HTML syntax, the document type declaration `<!DOCTYPE html>` is required, whose only purpose is to ensure that the document is rendered in standards mode.
- HTML generators that cannot output the short document type declaration above may use the document type declaration `<!DOCTYPE html SYSTEM "about:legacy-compat">` instead.
- See: The DOCTYPE

HTML Syntax: Case-insensitivity

Element and attribute names are case-insensitive:

- Element and attribute names, even those for foreign elements, may be written with any mix of lower- and uppercase letters that are an ASCII case-insensitive match for the name of the element/attribute.
- There must never be two or more attributes on the same start tag whose names are an ASCII case-insensitive match for each other.

HTML Syntax: Special Characters

- Element text must not contain the `'<'` character or an ambiguous `'&'` character.
  - The`script`, `style`, `textarea`, and `title` elements are exceptions.
- Attribute values must not contain an ambiguous `'&'` character.
- Ambiguous ampersand (`'&'`):
  - An `'&'` character that is followed by one or more ASCII alphanumerics, followed by a `';'`, where these characters do not match any of the named character references defined by the standard (e.g., `&nosuchchar;`).
    - See: Named character references

HTML Syntax: Unquoted Attribute Value Syntax

- If an attribute value other than the empty string does not contain any literal whitespace character, it can be specified by omitting the attribute value delimiters.
- For example, the following are equivalent:

```
<img src="w3c.png" width="120" height="120"
  alt="W3C logo">
<img src=w3c.png width=120 height=120 alt="W3C logo">
```

HTML Syntax: Boolean Attributes

- Several of attributes are boolean attributes.
- The presence of a boolean attribute on an element represents the true value, and the absence of the attribute represents the false value.
- If the attribute is present, its value must either be the empty string or a value that is a case-insensitive match for the attribute's canonical name, with no leading or trailing white space.
- For example, the following are equivalent:

```
<input type=checkbox checked name=agree disabled>
<input type=checkbox checked=checked name=agree
  disabled=disabled>
<input type='checkbox' checked='' name="agree"
  disabled="">
```

HTML Syntax: Void Elements

- Void elements only have a start tag; end tags must not be specified for them.
- Examples: `br`, `img`, `input`, `link`, `meta`

HTML Syntax: Optional Tags (1)

- The start and end tags of certain elements can be omitted.
- Omitting an element's start tag in the situations described here does not mean the element is not present (it is implied, but it is still there)!
  - For example, an HTML document always has a root `html` element, even if the string `<html>` doesn't appear anywhere in the markup.
- See: Optional tags

HTML Syntax: Optional Tags (2)

- An `li` element's end tag may be omitted if another `li` element immediately follows it or if there is no more content in the parent element.
  - For example, the following are equivalent:
    - ```
      <ul>
        <li>Apple</li>
        <li>Banana</li>
        <li>Cherry</li>
      </ul>
      ```
    - ```
      <ul>
        <li>Apple
        <li>Banana
        <li>Cherry
      </ul>
      ```

HTML Syntax: Optional Tags (3)

- An `html` element's start tag may be omitted if the first thing inside the element is not a comment.
- An `html` element's end tag may be omitted if a comment does not immediately follow the element.
- ...

HTML Syntax: Optional Tags (4)

- If whitespace between elements is not significant, the following are equivalent:
  - ```
    <!DOCTYPE html>
    <html>
      <head>
        <title>Example Page</title>
      </head>
      <body>
        Hello, World!
      </body>
    </html>
    ```
  - ```
    <!DOCTYPE html>
    <title>Example Page</title>
    <p>Hello, World!</p>
    ```

HTML Syntax: Foreign Elements

- Foreign elements must either have a start tag and an end tag, or a start tag that is marked as self-closing, in which case they must not have an end tag.
- For example, the following SVG elements are equivalent:

```
<circle cx="50" cy="50" r="50"></circle>
<circle cx="50" cy="50" r="50"/>
```

HTML Syntax: XML Features

- Empty-element tags, referred to as self-closing tags in HTML, are available only for foreign elements.
- Namespace declarations are not supported, even in foreign elements.
- CDATA sections can only be used in foreign content (MathML or SVG).

XML Syntax: Example

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Example Page</title>
    <link rel="stylesheet" href="style.css"/>
  </head>
  <body>
    <p>Visit W3C's website by clicking on the logo:</p>
    <a href="https://www.w3.org/">
      <img src="w3c.png" width="120" height="120"
        alt="W3C logo"/>
    </a>
  </body>
</html>
```

XML Syntax: DOCTYPE

- In the XML syntax, any document type declaration can be used, but it is not required.
- Documents transmitted with the `application/xhtml+xml` media type are always rendered in standards mode.
- See: Writing documents in the XML syntax

DOM (1)

- DOM stands for Document Object Model.
- It is an API for accessing and manipulating documents; in particular, HTML and XML documents.
- Current standard: DOM Living Standard (WHATWG)

DOM (2)

DOM tree:

- A DOM tree is an in-memory representation of a document.
- It is a tree structure that is made up of nodes of various types.
- Major node types:
  - `Attr`
  - `CDATASection`
  - `Comment`
  - `Document`
  - `DocumentType`
  - `Element`
  - `ProcessingInstruction`
  - `Text`

DOM (3)

Example:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>Sample Page</title>
 </head>
 <body>
  <p>Hello, World!</p>
  <!-- This is a comment -->
 </body>
</html>
```

```
├ DOCTYPE: html
└ html lang="en"
  ├ head
  │ ├ #text: ⏎␣␣
  │ ├ title
  │ │ └ #text: Sample Page
  │ └ #text: ⏎␣
  ├ #text: ⏎␣
  └ body
    ├ #text: ⏎␣␣
    ├ p
    │ └ #text: Hello, World!
    ├ #text: ⏎␣␣
    ├ #comment: This is a comment
    └ #text: ⏎␣⏎
```

DOM (4)

- Each node is represented by an object with an API so that it can be manipulated.
- DOM interfaces are described in Web IDL.

DOM (5)

Web IDL:

- Web IDL is an interface definition language that can be used to describe interfaces that are intended to be implemented in web browsers.
- Current standard: Web IDL Living Standard (WHATWG)
- Example of use: `Node` interface

DOM (6)

- The HTML specification defines additional interfaces that extend DOM interfaces for representing HTML elements.
- Examples:
  - `meta` element
  - `img` element

DOM (7)

- The DOM is not just an API; the conformance criteria of HTML implementations are defined in terms of operations on the DOM.
  - Example: The `Navigator` object
- The specifications are mostly phrased in terms of DOM trees, instead of markup.

DOM (8)

- A DOM tree can be manipulated from scripts in the page.
- Example:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>DOM Example</title>
  </head>
  <body>
    <p>User agent: <span id="ua"></span></p>
    <script>
      document.getElementById('ua').innerHTML = navigator.userAgent;
    </script>
  </body>
</html>
```

DOM (9)

- Tools:
  - Live DOM Viewer
- Further information:
  - MDN Web Docs – Document Object Model (DOM)
  - The Modern JavaScript Tutorial – DOM tree

Working with the DOM (1)

Working with the window:

```
// Querying the properties of the window:
console.log(window.location);
console.log(window.origin);
console.log(window.navigator.userAgent);
console.log(window.devicePixelRatio);
console.log(window.innerWidth, window.innerHeight);

// Scrolling the window:
window.scrollTo(0, 0);
window.scrollTo({top: 0, left: 0, behavior: "smooth"});
```

Working with the DOM (2)

Working with document metadata:

```
// Querying the media type of the document:
console.log(document.contentType);

// Querying the base URL of the document:
console.log(document.baseURI);

// Querying the title of the document:
console.log(document.title);

// Setting the title of the document:
document.title = 'Mastering the DOM';
```

Working with the DOM (3)

Working with elements:

```
// Querying the content of an element:
console.log(document.querySelector('h1').textContent);

// Modifying the content of an element:
document.getElementById('now').textContent = new Date().toLocaleString();
document.querySelector('footer').innerHTML =
  "<p><small>This content is in the public domain.</small></p>";
```

Working with the DOM (4)

Working with attributes:

```
// Querying the value of an attribute:
console.log(document.documentElement.lang)
console.log(document.querySelector('img#logo').alt);

// Setting the value of an attribute:
document.querySelector('a').target = 'blank_';
document.querySelectorAll('a')
  .forEach((element) => element.target = 'blank_');

// Querying the classes of an element:
console.log(document.body.className);

// Modifying the classes of an element:
document.getElementById('copyright').classList.add('important');
```

Working with the DOM (5)

```
// Setting values of CSS properties:
document.body.style.backgroundColor = 'aliceblue';
document.body.style.margin = '2rem';
document.body.style.marginTop = '1em';
document.getElementById('copyright').style.fontStyle = 'italic';

// Querying the computed value of a CSS property:
const fontSize = window.getComputedStyle(document.body).fontSize;
console.log(`Computed font size of body: ${fontSize}`);
```

Working with the DOM (6)

Hiding and showing an element:

```
<aside>
  <div id="help">
    <!-- ... -->
  </div>
  <button onclick="toggleHelp()">Hide/Show Help</button>
</aside>
<script>
  function toggleHelp() {
    const element = document.getElementById('help');
    element.hidden = !element.hidden;
  }
</script>
```

However, note that onevent attributes are not recommended to be used!

Working with the DOM (7)

Hiding and showing an element:

```
<aside>
  <div id="help">
    <!-- ... -->
  </div>
  <button id="toggle-help-btn">Hide/Show Help</button>
</aside>
<script>
  function toggleHelp(event) {
    const element = document.getElementById('help');
    element.hidden = !element.hidden;
  }
  document.getElementById('toggle-help-btn')
      .addEventListener('click', toggleHelp);
</script>
```

Working with the DOM (8)

Creating elements dynamically:

```
<div id="container"></div>
<button id="add-item-btn">Add Item</button>
<script>
  function addItem(event) {
    const container = document.getElementById('container');
    const item = document.createElement('div');
    item.className = 'item';
    container.appendChild(item);
  }
  document.getElementById('add-item-btn')
    .addEventListener('click', addItem);
</script>
```

Working with the DOM (9)

Removing elements dynamically:

```
<div id="container"></div>
<button id="remove-children-btn">Remove Children</button>
<script>
  function removeChildren(event) {
    const container = document.getElementById('container');
    while (container.firstChild) {
      container.removeChild(container.firstChild);
    }
  }
  document.getElementById('remove-children-btn')
    .addEventListener('click', removeChildren);
</script>
```

Comparing the DOM, the HTML Syntax, and the XML Syntax

The DOM, the HTML syntax, and the XML syntax cannot all represent the same content:

	HTML syntax	XML syntax	DOM
Namespaces	no	yes	yes
`noscript`	yes	no	no
The string `-->` in comments	no	no	yes

Error Handling (1)

- Unlike previous versions of HTML, the current specification defines in some detail the required processing for invalid documents as well as valid documents.
- Web browsers are very tolerant of errors; they automatically fix invalid content.
  - However, this applies to documents in the HTML syntax!
- See:
  - HTML Standard – Parsing HTML documents
    - An introduction to error handling and strange cases in the parser

Error Handling (2)

- Trying out the examples:
  - Examine the DOM trees in either the browser DevTools or the Live DOM Viewer!

Error Handling (3)

Example:

```
<html>
<body><b>How is <i>it</b> rendered?</i></body>
</html>
```

The invalid document is parsed into a DOM tree that is the same as that of the following valid document:

```
<html>
<head></head>
<body><b>How is <i>it</i></b><i> rendered?</i></body>
</html>
```

Error Handling (4)

Example:

```
<html>
  <foo>
  <p bar class=>Does it <b>work,<i> or</b> not</i>?
  <
  </body>
</html>
```

HTML APIs

- HTML as a language is not just about tags and elements.
- Modern web pages utilize many APIs defined either in the HTML Standard or in other standards developed by the WHATWG.
  - APIs defined as part of the HTML Standard include:
    - Canvas API
    - History API
    - Web Storage API
  - APIs defined in separate specifications:
    - Fetch API
    - Fullscreen API
    - WebSockets

Web Developer Tools in Browsers

- Major desktop browsers come with built-in web developer tools often referred to as DevTools.
- Documentation:
  - Chromium, Google Chrome, Opera: Chrome DevTools
  - Firefox: Firefox DevTools User Docs
  - Safari: Web development tools
  - Chromium-based Edge: Microsoft Edge DevTools documentation

Developer Editions of Browsers

- Google Chrome: Google Chrome for developers
- Firefox: Firefox Developer Edition

Editing HTML: Editors

Free and open source software:

- Visual Studio Code (platform: Linux, macOS, Windows; license: MIT License) https://code.visualstudio.com/ https://github.com/Microsoft/vscode
  - See: https://code.visualstudio.com/docs/languages/html
  - Recommended extensions:
    - Live Preview https://marketplace.visualstudio.com/items?itemName=ms-vscode.live-server https://github.com/microsoft/vscode-livepreview

Editing HTML: Emmet (1)

- A set of text editor plugins for boosting HTML and CSS code writing.
- Written in: JavaScript
- License: MIT License
- Website: https://emmet.io/
- Repository: https://github.com/emmetio/emmet

Editing HTML: Emmet (2)

- Available for many text editors including Eclipse, NetBeans, Notepad++, Visual Studio Code, and WebStorm.
  - https://emmet.io/download/
  - Emmet in Visual Studio Code
- Documentation: https://docs.emmet.io/
  - Customization: https://docs.emmet.io/customization/

Editing HTML: Emmet (3)

- Emmet provides abbreviations that are expanded to HTML or CSS code.
- The syntax of abbreviations is based on the syntax of CSS selectors.

Editing HTML: Emmet (4)

Examples of Emmet abbreviations:

`ul>li*3`

is expanded to:

```
<ul>
  <li></li>
  <li></li>
  <li></li>
</ul>
```

Editing HTML: Emmet (5)

Examples of Emmet abbreviations:

`section.chapter>h1{Introduction}+p`

is expanded to:

```
<section class="chapter">
  <h1>Introduction</h1>
  <p></p>
</section>
```

Editing HTML: Emmet (6)

Examples of Emmet abbreviations:

```
ul>li*5>a[href=#chapter$]{Chapter $}
```

is expanded to:

```
<ul>
  <li><a href="#chapter1">Chapter 1</a></li>
  <li><a href="#chapter2">Chapter 2</a></li>
  <li><a href="#chapter3">Chapter 3</a></li>
  <li><a href="#chapter4">Chapter 4</a></li>
  <li><a href="#chapter5">Chapter 5</a></li>
</ul>
```

HTML Tools: Validators (1)

- **Conformance checkers** (also known as **validators**) are software tools that check whether a given HTML document conforms to the syntax, structural, and semantic requirements of the HTML standard.

HTML Tools: Validators (2)

Nu Html Checker: a conformance checker that can be used either from the command line or online via a web interface

- Written in: Java
- License: MIT License
- Website: https://validator.github.io/validator/
- Repository: https://github.com/validator/validator
- An instance operated by the W3C is available at https://validator.w3.org/nu/

HTML Tools: Tidying Tools

Tidy: a command-line tool to correct and clean up HTML documents

- Platform: Linux, macOS, Windows
- Written in: C
- License: Tidy License
- Website: https://www.html-tidy.org/
- Repository: https://github.com/htacg/tidy-html5

HTML Tools: Web Servers

http-server: a simple, zero-configuration command-line static HTTP server

- Written in: JavaScript
- License: MIT License
- Repository: https://github.com/http-party/http-server
- Installation and usage:

```
$ npx http-server -o index.html

$ npm install -g http-server
$ http-server -o index.html
```

- The document is available at http://localhost:8080/index.html.

HTML Tools: Hosting Services

Surge: free static HTML publishing from the command line

- Website: https://surge.sh/
- Documentation: https://surge.sh/help/
- Installation and usage (deploying the current directory):

```
$ npm install --global surge
$ surge --domain example.surge.sh
```

- The deployed website is available at https://example.surge.sh/.