





Reading Sample

This chapter describes the displayable elements of HTML that you can use between `<body>` and `</body>`. It discusses how to create a single web page using HTML and mark it up with the appropriate elements. Topics covered include splitting an HTML document into separate and meaningful sections, using headings in a predetermined order, implementing a header and footer, splitting and grouping text content, tagging semantic elements, and using and displaying unordered and ordered lists.

-  **"The Visible Part of an HTML Document"**
-  **Contents**
-  **Index**
-  **The Author**

Jürgen Wolf

HTML and CSS

The Comprehensive Guide

814 pages | 04/2023 | \$59.95 | ISBN 978-1-4932-2422-7

 www.rheinwerk-computing.com/5695

Chapter 4

The Visible Part of an HTML Document

This chapter describes the displayable elements of HTML that you can use between `<body>` and `</body>`. For designing or laying out websites, you should use CSS instead. Before you learn how to make a website more beautiful, you need to have the basic knowledge of how to create a single web page using HTML and mark it up with the appropriate elements.

Even if you’ve already created web pages in HTML 4.01 (or even earlier) and are already familiar with the handling of HTML elements, it’s worth working through this chapter because semantic elements have been added with the current HTML and many existing elements have been given a different semantic meaning.

Here’s what you’ll learn in this chapter:

- Splitting an HTML document into separate and meaningful sections with new HTML elements such as `<section>`, `<article>`, `<aside>`, or `<nav>`
- Using headings in a certain order and implementing a header and/or footer with the new `<header>` and `<footer>` elements
- Splitting and grouping text content with HTML elements
- Semantic tagging of text such as single letters, words, or parts of sentences with HTML elements
- Using and displaying unordered and ordered lists via `` and ``

4.1 HTML Elements for Structuring Pages

In this chapter, you’ll learn about the various HTML elements that you can use to divide a web page into useful sections. If you’ve used HTML 4.01 so far, you’ll find many new elements here, as the current HTML also introduces a new content model to combat the rampant use of `div` elements with `class` attributes.

HTML Element	Meaning
<code><body></code>	Displayable content section of the HTML document
<code><section></code>	Subdivision of the HTML document into different sections

Table 4.1 Quick Overview of the Section Elements Covered Here

HTML Element	Meaning
<article>	Subdivision of content into a self-contained topic-specific block
<aside>	Marginal information of a content such as a sidebar or for additional information about an article
<nav>	Element used to mark up navigation(s) such as a sitemap or the main navigation of the website
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	Headings of the first through sixth order
<header>	Header of a content
<footer>	Footer of a content
<address>	Contact information for the author of the content

Table 4.1 Quick Overview of the Section Elements Covered Here (Cont.)

4.1.1 Using <body>: The Displayable Content Section of an HTML Document

Everything you write between the opening <body> tag and the closing </body> tag is referred to as the *HTML document body*. Between <body> and </body>, you can write all HTML elements, such as text, hypertext links, images, tables, and lists, to define the structure of the web page. All elements written between <body> and </body> are rendered by the web browser and displayed accordingly.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Title of the document</title>
    <meta charset="UTF-8">
  </head>
  <body>
    This is the content of the document, which is to be
    rendered and displayed by the web browser.
  </body>
</html>
```

4.1.2 Introducing the Section Elements of HTML

The following sections introduce the section elements of HTML, that is, <section>, <article>, <aside>, and <nav>. If you're perhaps just getting into HTML, using section elements is still a bit confusing or disappointing at first because they change almost

nothing visually. Primarily, these elements only serve to divide the content into semantic (i.e., meaningful) areas.

Even if these new elements don't seem to make sense to you yet, just remember that they aren't of interest to the normal user of the website, but are mainly used to give meaning to the content, which is particularly useful for the developer, the search engines, and the screen readers.

Dividing Content into Topic-Based Sections Using <section>

The <section> element allows you to divide the content of a document into topic-based sections. This is helpful, for example, if you want to divide a document into individual chapters or even subchapters—just like this book was divided into individual sections. Even on an ordinary homepage, you can use this element to create individual content and sense sections, such as a section with the description about the owner of the website, another section with news, and one with contact information. Here's a simple example, the result of which you can see in Figure 4.1:

```
...
<body>
  <section>
    <h1>Chapter 1</h1>
    <p>The first chapter</p>
  </section>
  <section>
    <h1>Chapter 2</h1>
    <p>The second chapter</p>
    <section>
      <h2>Chapter 2.1</h2>
      <p>A subchapter of Ch. 2</p>
    </section>
  </section>
</body>
...
```

Listing 4.1 /examples/chapter004/4_1_3/index.html

In this example, the <section> element has been used to divide the document into meaningful sections—in this case, Chapter 1 and Chapter 2—with each chapter consisting of a heading <h1> and paragraph text <p>. Furthermore, it's possible to nest <section> elements, as shown within the <section> element of Chapter 2, Section 2.1.

Chapter 1

The first chapter

section 705.6 × 65.51

Chapter 2

The second chapter

Chapter 2.1

A subchapter of Ch. 2

Figure 4.1 Between `<section>` and `</section>`, You Can Divide the Content of a Document into Meaningful and Logical Units

Dividing Content into a Self-Contained Block Using `<article>`

You should use the `article` element to summarize a piece of content in a self-contained topic-specific block. The `article` element is in itself quite similar to the `<section>` element, which you use to divide the content into meaningful sections. However, it's recommended that you use the `article` element for a standalone composition, which would be ideal for individual news items, blog or forum entries, or comments on a blog post or news, for example.

Here's an example of an HTML code snippet that shows you what such a blog entry with the `article` element could look like. The result is shown in Figure 4.2.

```
...
<body>
<h1>My Blog</h1>
<p>Latest reports on HTML</p>
<article>
  <header>
    <h2>New HTML elements on the horizon</h2>
  </header>
  <p>Published on <time>2023-05-05</time></p>
  <p>As already suspected ...</p>
  <footer>
    <a href="comments.html">View comments ...</a>
  </footer>
</article>
</body>
...
```

Listing 4.2 `/examples/chapter004/4_1_4/index.html`

My Blog

article 705.6 × 134.31 L

New HTML elements on the horizon

Published on 2023-05-05

As already suspected ...

[View comments ...](#)

Figure 4.2 The Example Shows a Meaningful and Logical `<article>` Composition of a Blog Entry

Everything between `<article>` and `</article>` is the composition of a self-contained block consisting of a heading, a timestamp, the actual content section, and a footer. It's up to you to decide which HTML elements you want to use to create such a composition with `<article>`, but the example shown here already makes sense semantically.

What to Use: `<article>` or `<section>`?

You're probably wondering which of the two elements you should use for a semantic separation of content because the two are somewhat similar in some respects. Nevertheless, the HTML specification also makes a differentiation here and recommends using `<article>` if certain semantics are to be used multiple times, as is the case with a news or blog entry. Thus, `<article>` is a self-contained block—a composition of repeatedly used content following the same pattern—whereas `<section>` is suitable for a separation into content sections, which should contribute to a better overview of the entire document.

Adding Content with Additional Information Using `<aside>`

With `<aside>`, you can usually supplement or expand content with additional information. Strictly speaking, you can use the `aside` element for two different semantic things: a sidebar or an additional piece of information (e.g., a citation) to a content item, for example, within an `article` element.

Referring to the `/examples/chapter004/4_1_4/index.html` example from the previous section, for example, you would use `<aside>` for a separate logical section in the document:

```
...
<body>
<h1>My Blog</h1>
```

```
<p>Latest reports on HTML</p>
<article>
...
</article>
<aside>
<h3>Partner websites</h3>
  <ul>
    <li><a href="#">Blog XY</a></li>
    <li><a href="#">Magazine X</a></li>
    <li><a href="#">Website Z</a></li>
  </ul>
</aside>
</body>
...
```

Listing 4.3 /examples/chapter004/4_1_5/index.html

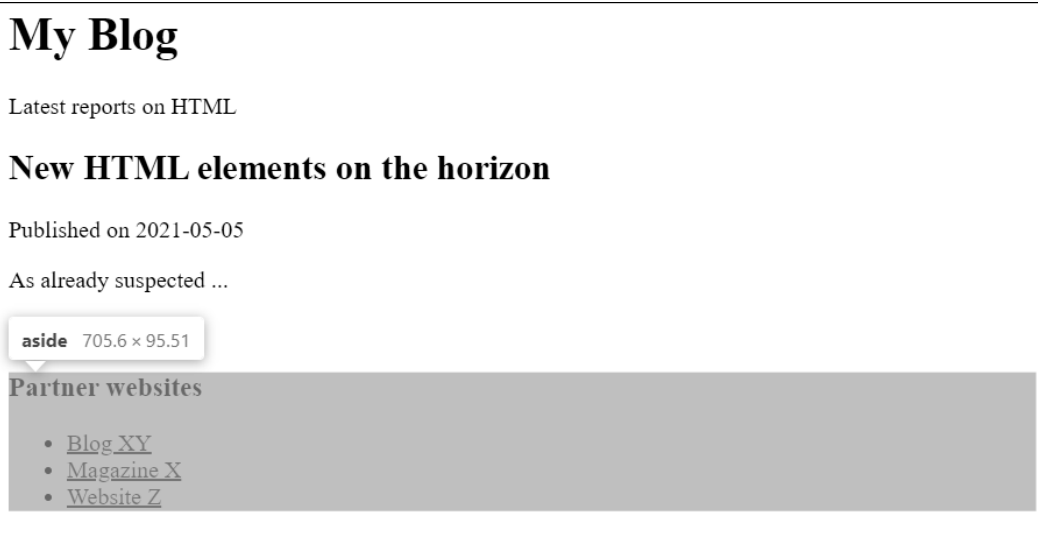


Figure 4.3 The <aside> Element Is Used as a Separate Logical Section in the HTML Document

Note

The # character in HTML is a reference to a jump mark in the same document, but it has no meaning yet in this example and was used instead of a real destination address.

In addition to the option just shown, using <aside> as a sidebar would also be suitable as additional information in the form of a quote or within an article element. In the example that contains the blog entry, it would be suitable within the article element for an entry with further links to the blog entry.

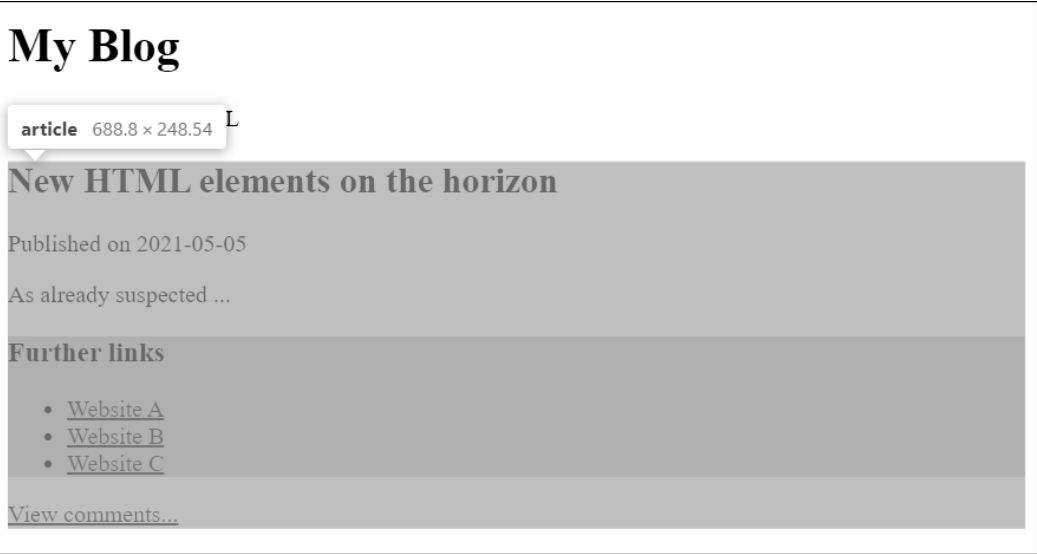


Figure 4.4 The <aside> Element (Colored Here) Was Noted as Additional Information inside an <article> Element

Let’s take a look at the following code snippet in this regard:

```
...
<body>
...
<article>
  <header>
    <h1>New HTML elements on the horizon</h1>
  </header>
  <p>Published on <time>2023-05-05</time></p>
  <p>As already suspected ...</p>
  <aside>
    <h3>Further links</h3>
    <ul>
      <li><a href="#">Website A</a></li>
      <li><a href="#">Website B</a></li>
      <li><a href="#">Website C</a></li>
    </ul>
  </aside>
  <footer>
    <a href="comments.html">View comments...</a>
  </footer>
</article>
<aside>
...
```

```
</aside>
</body>
...
```

Listing 4.4 /examples/chapter004/4_1_5/index2.html

Declaring Content as a Page Navigation Bar Using <nav>

As you might guess from its name, the `nav` element enables you to divide navigation elements into blocks. We’re not talking about web link collections here, but about a list of links for a sitemap or the main navigation of your own website. Like the `aside` element, you can use the `nav` element for its own section or within another HTML element to combine a group of links into a block.

To use the blog entry again as an example, the `nav` element would be suitable for summarizing the main navigation or the list of related links from similar articles within the same web page. In any case, you should use the `nav` element for entire blocks of links. The following code snippet demonstrates the `nav` element in a small theoretical blog:

```
...
<body>
  <nav>
    <a href="#">Blog</a> |
    <a href="#">Links</a> |
    <a href="#">About me</a> |
    <a href="#">Legal Notes</a>
  </nav>
  <h1>My Blog</h1>
  <p>latest reports on HTML</p>
  <article>
...
    <aside>
      <h3>Similar articles</h3>
      <nav>
        <ul>
          <li><a href="#">HTML6 will not exist</a></li>.
          <li><a href="#">W3C and WHATWG agree</a></li>
          <li><a href="#">What comes after the Living Standard?</a></li>
        </ul>
      </nav>
    </aside>
...
  </article>
  <aside>
    <h3>Sitemap</h3>
    <nav>
```

```
<ul>
  <li><a href="#">Blog</a>
    <ul>
      <li><a href="#">HTML</a></li>
      <li><a href="#">CSS</a></li>
    </ul>
  </li>
  <li><a href="#">Links</a></li>
  <li><a href="#">About me</a>
    <ul>
      <li><a href="#">Bio</a></li>
      <li><a href="#">Portfolio</a></li>
    </ul>
  </li>
  <li><a href="#">Legal Notes</a></li>
</ul>
</nav>
</aside>
</body>
...
```

Listing 4.5 /examples/chapter004/4_1_6/index.html



Figure 4.5 The `<nav>` Element (Colored Here) Can Be Used to Divide a Separate (Navigation) Section or to Group Blocks of Links within Other HTML Elements

In the first example, `<nav>` was used to define a main navigation as a separate section of the HTML document. In the second example, the `nav` element was used to link a block of links to similar articles on the same web page. In the last example, a sitemap of the web page was summarized via the `nav` element.

In addition, the last two examples were grouped within `<aside>` and `</aside>`. In them, bulleted lists (`ul` and `li` elements) were used within the `nav` element.

Using `<nav>` Only for Main Navigation?

The specification suggests using the `nav` element specifically for the main navigation. That doesn't include external additional links or affiliate links to external websites. Likewise, it's not recommended to put legal stuff such as copyright, contact information, and legal notes in the `nav` section; instead, use the `footer` section for that purpose (see Section 4.1.4).

In the `/examples/chapter004/4_1_6/index.html` example, this means you only have two main navigation points with `Blog` and `Links` within the `nav` element and would write `About me` and `Legal Notes` outside of it (e.g., in the footer). I don't see any point in separating this because there's no difference between the first two links (**Blog** and **Links**) and the other two links (**About me** and **Legal Notes**) as both are linked to different websites, and both belong to the internal website in this case. So, it's up to you to what extent you want to follow these recommendations. In any case, you should refrain from using the `nav` element for external links to third-party websites and, if possible, use it only selectively and sensibly on your own website.

4.1.3 Using Headings with the HTML Elements from `<h1>` to `<h6>`

The HTML element for headings of a certain order is `<h1>` to `<h6>`. The number (1 to 6) represents the heading level. Thus, everything you write between `<h1>` and `</h1>` is used as a top-level heading, everything between `<h2>` and `</h2>` belongs to a second-level heading, and so on down to the lowest level with `<h6>` and `</h6>` as a sixth-level heading.

The HTML elements `<h1>` through `<h6>` should not be misused to emphasize a text, but rather to define the content structure of a document. Consider the following HTML structure:

```
...
<h1>Heading 1</h1>
<h2>Heading 1.1</h2>
<h3>Heading 1.1.1</h3>
<h2>Heading 1.2</h2>
<h2>Heading 1.3</h2>
<h3>Heading 1.3.1</h3>
```

```
<h1>Heading 2</h1>
...
```

Based on this sequence of headings, the following content structure (or *document outline*) will be mapped:

- 1. Heading 1
 - 1.1. Heading 1.1
 - 1.1.1. Heading 1.1.1
 - 1.2. Heading 1.2
 - 1.3. Heading 1.3
 - 1.3.1. Heading 1.3.1
- 2. Heading 2

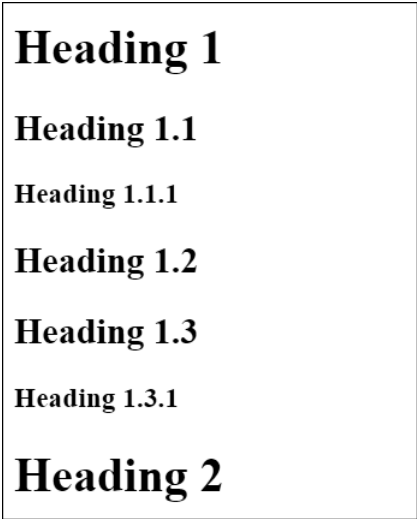


Figure 4.6 This Is What the Web Browser Will Make of It

What Happens to the Headings in the Section Elements?

You're probably wondering what happens to the content structure of headings when you use the section elements from Section 4.1.2. That question is well justified. If you use the `<section>`, `<article>`, `<aside>`, or `<nav>` section elements, the content structure of the headings will also be affected. Within each new section element, the heading level count starts from the beginning, but always at a lower hierarchy level. The following HTML code illustrates this:

```
...
<body>
  <h1>My Blog</h1>
  <p>A simple blog ...</p>
  <section>
```

```
<h1>News on HTML</h1>
<article>
  <h1>A preview of the new HTML elements</h1>
  <p>It looks like ...</p>
</article>
</section>
<section>
  <h1>News on CSS</h1>
  <article>
    <h1>New Styles at Last</h1>
    <p>After a long time of development ...</p>
  </article>
</section>
</body>
...
```

Listing 4.6 /examples/chapter004/4_1_7/index.html

Here, five <h1> headings of the first order were used. If you look at the HTML code, you can see several sections. Next to the top section with <body>, you can find two additional <section> elements, each of which contains an <article> element in which headings of the first order have also been defined.

This is a blog that’s been divided into two content sections with <section> containing the topics HTML and CSS. Within these sections, you can find the news articles included within <article>.



Figure 4.7 All Headings with <h1> Are Adjusted and Output Corresponds to the Section due to the Section Elements of HTML That Are Based on the Outline Algorithm

Due to the use of the new HTML elements <article> and <section>, the following content structure (or document outline) results:

- 1. My Blog
 - 1. News on HTML
 - 1. A preview of the new HTML elements
 - 2. News on CSS
 - 1. New Styles at Last

Document Outline for Advanced Users

The term *outline* or *document outline* refers to the structure of the document, which can be generated and represented by the headings, among other things—as in the case of the table of contents of this book, for example. The document outline can be quite useful. For example, the web browser might offer you a table of contents, letting you jump from one heading to another. Search engines can also use such a table of contents to create better page previews or even improve search results. Screen reader users probably have the biggest advantage here because they can be guided through deeply nested hierarchies and sections.

In Figure 4.8, you can see the JavaScript HTML5 outliner (h5o) to test the document outline during execution. Here, the document outline is displayed in the upper-right corner, and you can jump to the individual headings via hypertext links.

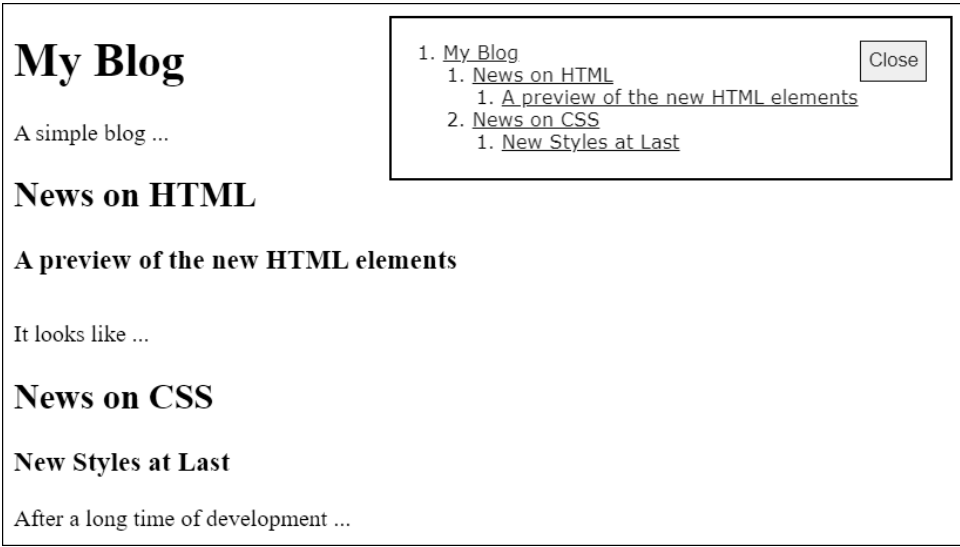


Figure 4.8 JavaScript h5o from Google during Execution

Section elements such as <section>, <article>, <aside>, and <nav> allow you to refine the document outline even more, as you’ve seen in the example, /examples/chapter004/4_1_7/index.html.

Even if not all web browsers support document outlines directly, it won't do any harm to pay attention to a proper outline of the HTML document because basically that's no extra work. Screen reader users will thank you for it, and search engine robots may reward you for it because a good document outline can improve a page index, which in turn could mean a higher ranking in search results. Besides, a neatly structured web page is easier to read than an unstructured one.

Keeping Track of the Document Outline

When the website becomes more extensive and the document contains many headings and perhaps different sections, it often isn't easy to keep track of whether the document outline still makes sense and is neatly structured with regard to its contents. Outlining tools that output the headline structure of the web page in the existing structure can assist you here. For example, Google offers the JavaScript h5o, mentioned in the previous section, at <https://h5o.github.io>. Alternatively, you can find an online service at <http://gsnedders.html5.org/outliner/>. Meanwhile, the validation checker at <https://validator.w3.org/nu/#textarea> also provides an outline option for HTML documents.

4.1.4 Creating a Header Using <header> and a Footer Using <footer>

The <header> and <footer> are two additional semantic HTML elements that you can use for implementing a header and footer in an HTML document. Like section elements, these elements initially have no visual effect on the HTML document apart from a line break. Again, these are initially just elements that you can use to give a piece of content a better and cleaner structure. The styling here is usually done via CSS. However, unlike section elements such as <section>, <article>, <aside>, or <nav>, these two elements don't affect the hierarchical structuring (or document outline) of the document.

You should use the header element for introductory elements such as a page heading, the name of the web page, or a navigation bar of the HTML document. There may well be other HTML elements between <header> and </header>. However, you mustn't nest any other header elements in it. Although it seems obvious, <header> doesn't necessarily have to be in the header, and you can use it more than once in the document.

Invalid Positions of <header>

A <header> tag mustn't be used inside a footer, address, or another header element.

The counterpart to the header element for the header section is the footer element for the footer or also the footer section, which also doesn't necessarily have to be the last element in the document. Useful content for the footer of a website is often legal

information, legal notes, or terms and conditions, but you can also use a sitemap or a special navigation bar here. You can't use any other <footer> tag within a footer

Here's an example that demonstrates the header and footer elements in a meaningful structure:

```
...
<body>
  <header>
    <hr /><small>Blog Version 1.0</small>
    <h1>My Blog</h1>
    <p>A simple blog...</p><hr />
  </header>
  <h2>News on HTML</h2>
  <article>
    <h3>A preview of the new HTML elements</h3>
    <p>It looks like ...</p>
  </article>
  <footer>
    <hr /><a href="#">Legal</a> |
    <a href="#">Legal Notes</a> |
    <a href="#">T&Cs</a> |
    <a href="#">About me</a><hr />
  </footer>
</body>
...
```

Listing 4.7 /examples/chapter004/4_1_8/index.html

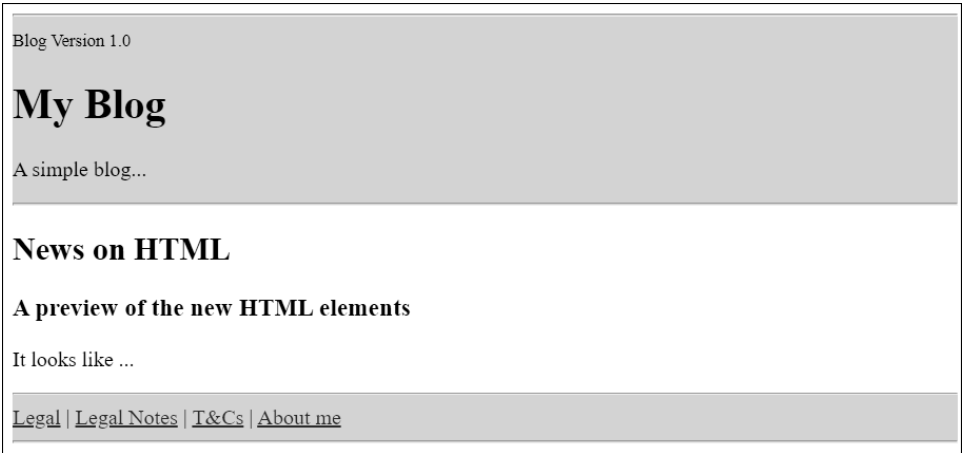


Figure 4.9 The Header and Footer with the <header> and <footer> Elements (Shown in Gray for Clarity)

Between `<header>` and `</header>`, you can find a summary of the entire information for the header section of a web page. In the example, this is the version of the blog, the headline, and a short description of the website. This is followed by the articles of the blog. Finally, the footer between `<footer>` and `</footer>` contains forwarding hyperlinks with legal information and so on.

4.1.5 Marking Contact Information Using `<address>`

You should use the `address` element only for contact information about the author of the HTML document or article. If the `address` element is used within the `body` element, it should only contain the contact information for the owner or author of the entire document or article. If the `address` element is positioned inside an `article` element, the contact information for the author of the document should be written there. Usually, the web browser displays this text in italics with a new line before and after the `address` element.

The best location for this contact information for the author, an organization, or the person responsible for the document or article is usually likely to be at the end of the article or at the end of the document (e.g., between `<footer>` and `</footer>`).

Here’s an example in which the `address` element was used for contact information about the author of an article at the end inside the footer element. You can see the example at execution in Figure 4.10.

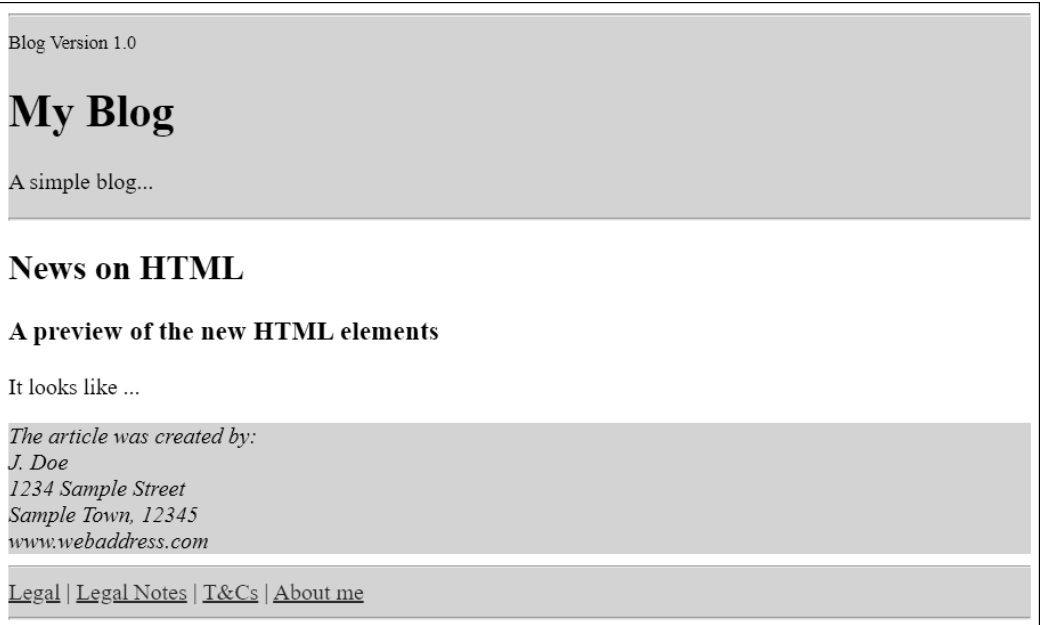


Figure 4.10 Contact Information for the Author of the Article Has Been Placed at the End of the Article between `<footer>` and `</footer>` Using the `<address>` Element

```
...
<article>
  <h3>A preview of the new HTML elements</h3>
  <p>It looks like ...</p>
  <footer>
    <address>The article was created by:<br>
      J. Doe<br>
      1234 Sample Street<br>
      Sample Town, 12345<br>
      www.webaddress.com
    </address>
  </footer>
</article>
...
```

Listing 4.8 /examples/chapter004/4_1_9/index.html

4.2 HTML Elements for Structuring Text

This section describes the HTML elements for grouping or structuring plain text content, such as paragraph text or a line break. This has nothing to do with dividing an HTML document into individual sections or areas. You’ve previously learned how to do that in Section 4.1.

HTML Element	Meaning
<code><p></code>	Text paragraph
<code>
</code>	Forcing a line break
<code><wbr></code>	Optional line break within a word
<code><hr></code>	Topic-based separation at the paragraph level
<code><blockquote></code>	Citation as a text paragraph
<code><div></code>	Defining a general section
<code><main></code>	Used for the main content area of a web page
<code><figure></code>	Grouping or summarizing content for separate description
<code><figcaption></code>	Labeling content grouped via the <code>figure</code> element
<code></code>	Unordered bulleted list
<code></code>	Ordered list (mostly numbered)

Table 4.2 Brief Overview of the Elements Covered Here for Grouping and Dividing Content

HTML Element	Meaning
	List element in a ul or ol list
<dl>	Creating a description list using dt and dd
<dt>	Expression to be described before the dd element
<dd>	Description that follows the dt element

Table 4.2 Brief Overview of the Elements Covered Here for Grouping and Dividing Content (Cont.)

4.2.1 Adding Text Paragraphs Using <p>

The `p` element (`p` = *paragraph*) is the classic element for text paragraphs in a longer continuous text. Anything you write here between the opening `<p>` and the closing `</p>` is treated as a text paragraph. Within such a text paragraph you can use images, videos, audio clips, or other text markup in addition to multiline body text. However, you can't use other group elements, headings (`<h1>` to `<h6>`), or section elements within `<p>` and `</p>`.

The following example demonstrates two slightly longer paragraph texts with the `p` element in use:

```
...
<body>
...
<h2>News on HTML</h2>
<article>
  <h3>A preview of the new HTML elements</h3>.
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Aenean commodo ligula eget dolor. Aenean massa.
    Cum sociis natoque penatibus et magnis dis parturient
    montes, nascetur ridiculus mus. Donec quam felis,
    ultricies nec, pellentesque eu, pretium quis, sem.
    Nulla consequat massa quis enim. Donec pede justo,
    fringilla vel, aliquet nec, vulputate eget, arcu. In
    enim justo, rhoncus ut, imperdiet a, venenatis vitae,
    justo.
  </p>
  <p>Nullam dictum felis eu pede mollis pretium. Integer
    tincidunt. Cras dapibus. Vivamus elementum semper
    nisi. Aenean vulputate eleifend tellus. Aenean leo
    ligula, porttitor eu, consequat vitae, eleifend ac,
    enim. Aliquam lorem ante, dapibus in, viverra quis,
    feugiat a, tellus.
```

```
    </p>
  </article>
  ...
</body>
...
```

Listing 4.9 /examples/chapter004/4_2_1/index.html

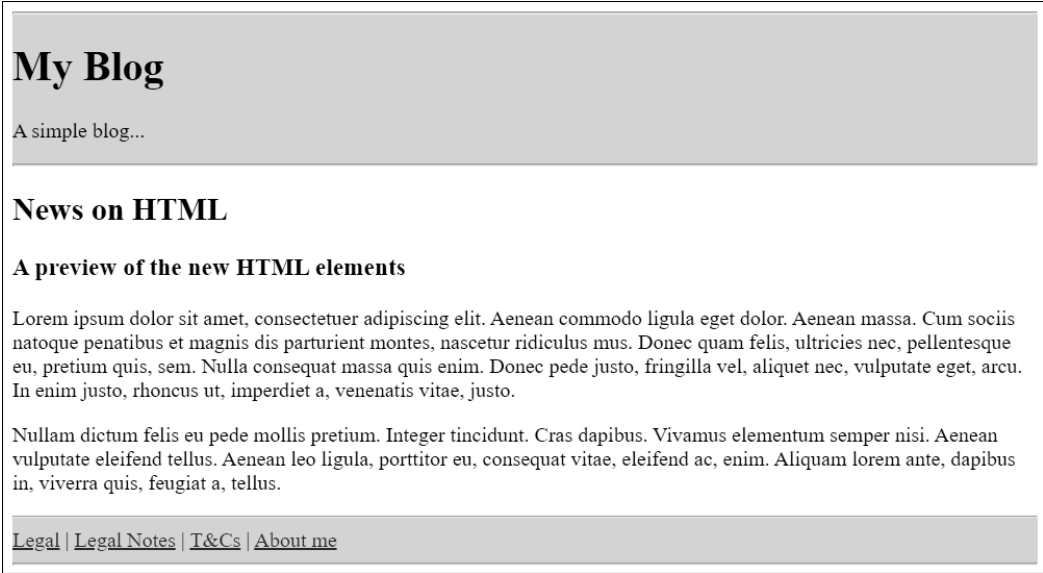


Figure 4.11 Two Paragraphs with Body Text between `<p>` and `</p>` Displayed in the Web Browser

Aligning and Formatting Paragraph Text Using CSS

Paragraph text with the `p` element can be formatted using CSS or CSS features.

4.2.2 Forcing Line Breaks Using

If you try to insert a line break or a space in the body text of the example just shown, `/examples/chapter004/4_2_1/index.html`, you'll notice that it doesn't work. The point at which the line break is supposed to be inserted is decided by the web browser based on a space that separates words. Nevertheless, you can also force a line break at a certain point in the text using `
` (`br` = *break*). `
` is a standalone tag. Even though you can use multiple line breaks simultaneously via `
`, you shouldn't overuse it for separating paragraphs.

The following example is commonly used to represent an address neatly by means of forced line breaks (see Figure 4.12):

```
...
Written by <a href="mailto:#">John</a><br>
<address>
    John Doe<br>
    Sample Town<br>
    www.address.com
</address>
...
```

Listing 4.10 /examples/chapter004/4 2 2/index.html



Figure 4.12 You Can Force Line Breaks via the
 Element

4.2.3 Adding Optional Line Breaks Using <wbr>

If, on the other hand, you need an optional line break that only occurs at a specific position when it's necessary for an optimal display in the web browser and to save the user from scrolling sideways, you can use the standalone `<wbr>` (or `<wbr />` in XHTML) tag for this (`wbr` = *word break*). `<wbr>` can be quite useful if you want to prevent the web browser from breaking a line in the wrong place. A simple example follows:

<p>Taumatawhakatangihangakoauauotamateaturipukakapikimaungahoronukupokaiwhenuakitanatahu</p>

Depending on how wide the display section is in the web browser, the long word can be wrapped only at the places where `<wbr>` was inserted.

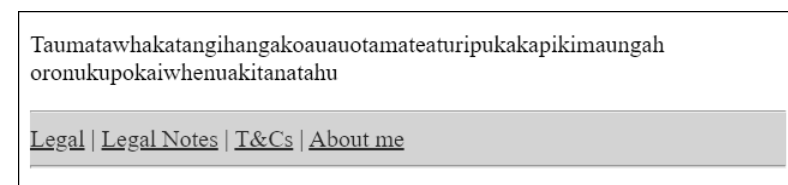


Figure 4.13 An Extremely Long Word Wrapped at a Position Suggested by `<wbr>`

Line Break via <wbr>

The optional line break via `<wbr>` was added to the standard HTML version at a later time, but it had been around since HTML 2.0. The Netscape web browser had introduced this element a long time ago, and other browser manufacturers had implemented it as well.

Nevertheless, this word separation is unattractive because the line gets broken without consideration of a grammatically correct separation. As an alternative, the *named entity* ­ is suitable for a conditional hyphen. Similar to <wbr>, this allows the web browser to separate the word at this point if necessary. Unlike <wbr>, the web browser adds a hyphen at the end of the wrapped word. The alternative example looks as follows:

<p>Taumatawhakatangi­
hangakoauauotamatea­
turipukakapikimaungah­
oronukupokaiwhen­
uakitanatahu</p>

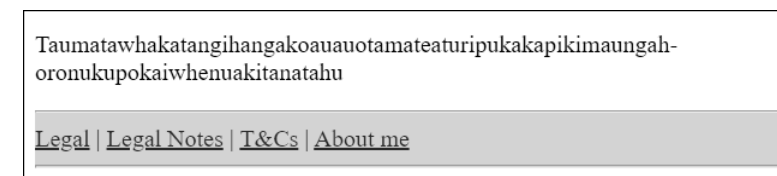


Figure 4.14 A Long Word Can Also Be Wrapped at the Position Suggested by “­” but It Also Adds a Separator, Unlike `<wbr>`

4.2.4 Forcing Spaces and Preventing Wrapping Using " ";

If you want to insert multiple spaces between two words, you can force this with the named character ` `; (` ` = *nonbreaking space*). Let's take a look at a simple example:

[illegible]

Between `word1` and `word2`, the named HTML entity character ` ` has been written four times, resulting in four spaces between these two words, which is also displayed by the web browser. In the example with `word3` and `word4` the same was tried by pressing the spacebar four times. Nevertheless, in this case, the web browser displays only one space between the words.

In addition to forcing a space, you can also use the named entity character ` `; to prevent a break between two words, which the web browser automatically performs when there is insufficient space at the end of a line. For example:

```
... word1&nbsp;word2 ...
```

By placing this ` ` between `word1` and `word2`, you can prevent the two words from being split between two different lines by the web browser if there's a lack of space. `word1` and `word2` thus stick together in the same line forever.

4.2.5 Adding a Topic-Based Separation Using `<hr>`

You can use `<hr>` to create a topic-based separation in an HTML document, for example, to separate content more clearly. However, even though `<hr>` is visualized as a separator in HTML by web browsers, the element is also to be treated as a semantic element and not a presentation element. For example, it isn't valid HTML to use the `hr` element between `<p>` and `</p>` or within a heading (`<h1>` to `<h6>`), even though web browsers are quite fault-tolerant about this.

The example shown in Figure 4.15 with a horizontal line can be found under `/examples/chapter004/4_2_5/index.html`. You'll see that a separator line also creates a paragraph.

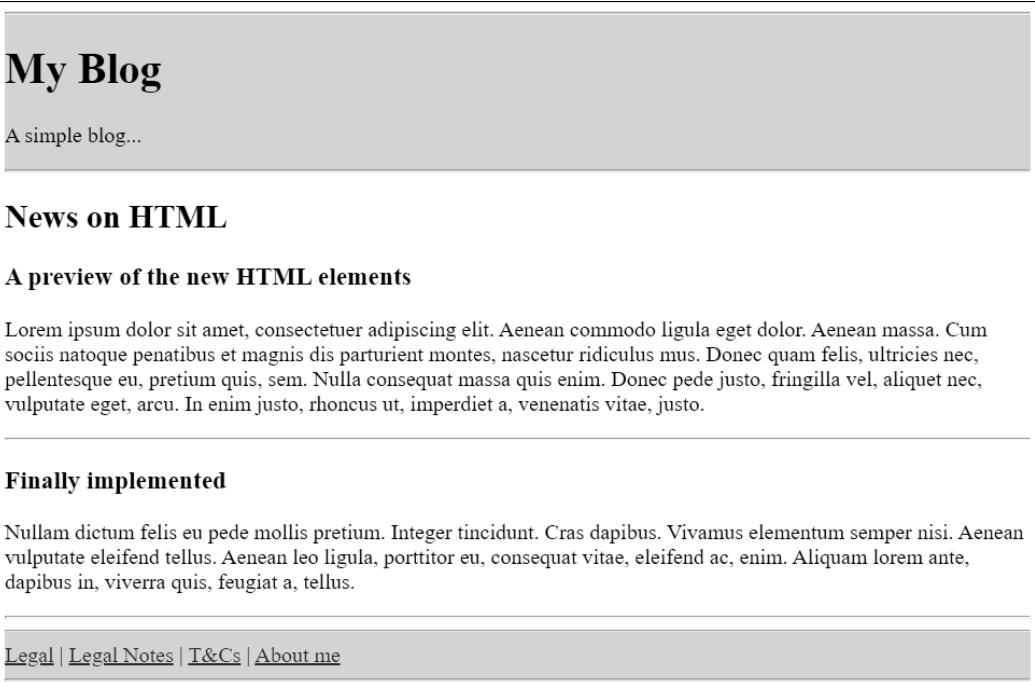


Figure 4.15 With `<hr>`, a Visual Topic-Based Separation Has Been Added as a Separator Line behind the Paragraph Text

4.2.6 Adding Paragraphs or Citations Using `<blockquote>`

Between `<blockquote>` and `</blockquote>`, you can quote a text from another source. Most web browsers indent the text in a new paragraph. Within such block quotes, you can use other HTML elements besides text.

The `blockquote` element contains `cite`, an HTML attribute that allows you to specify the source of the citation. With regard to books, this can also be a link to the corresponding book page or to a store where this book can be purchased. Unfortunately, no web browser provides the option to somehow make this source visible or to call the corresponding URL yet. So, to be on the safe side, you should add the source, as I did in the following example; in Figure 4.16, you can see the display in the web browser.

```
...
<blockquote cite="http://www.blindtextgenerator.com/">
  Nulla consequat massa quis enim. Donec pede justo,
  fringilla vel, aliquet nec, vulputate eget, arcu. In enim
  justo, rhoncus ut, imperdiet a, venenatis vitae, justo.
  <small> - http://www.blindtextgenerator.com/ - </small>
</blockquote>
...
```

Listing 4.11 `/examples/chapter004/4_2_6/index.html`

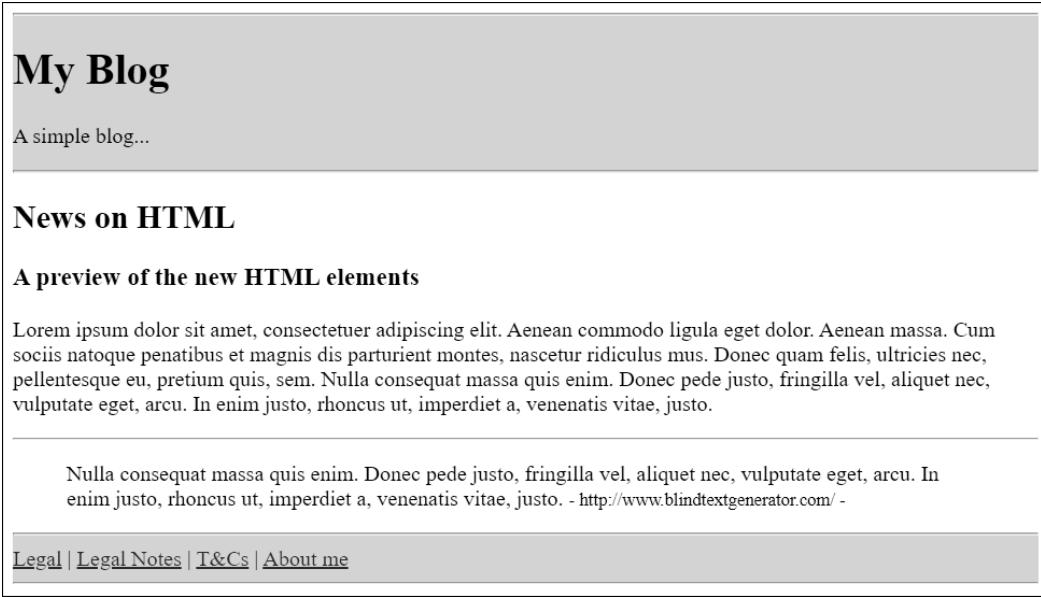


Figure 4.16 Text Quoted between `<blockquote>` and `</blockquote>` from the `www.blindtext-generator.com` Website

4.2.7 Defining a General Section Using <div>

Between <div> and </div> (div = *division*), you can define a general section, which at first usually does nothing but create a new line. This div element doesn't have any meaning until CSS comes into play, which is the main use of <div>: defining layout sections. In the following example, the HTML attribute class was used, which you can use to assign the div elements to a class that you can later select with CSS (using a selector) and visually customize or style. Here's a familiar example that demonstrates such an application in use:

```
...
<body>
  <div class="header">
    <hr />
    <h1>My Blog</h1>
    <p>A simple blog ...</p>
    <hr />
  </div>
  <h2>News on HTML</h2>
  <div class="article">
    <h3>A preview of the new HTML elements</h3>
    <p>Lorem ipsum dolor ...</p>
  </div>
  <div class="footer">
    <hr />
    <a href="#">Legal</a> |
    <a href="#">Legal Notes</a> |
    <a href="#">T&Cs</a> |
    <a href="#">About me</a>
  </div>
</body>
...
```

Listing 4.12 /examples/chapter004/4_2_7/index.html

For such examples, you should prefer semantic elements such as <header>, <footer>, <article>, <nav>, and so on instead of the div element. Therefore, you should use the div element only if no other suitable HTML element is available. You can find more information about this in greater detail in Section 4.3. In regard to the /examples/chapter004/4_2_7/index.html example, you should, as previously described in the book, use the HTML elements <header>, <article>, and <footer> that have been newly introduced in HTML instead of the <div class="header">, <div class="article">, and <div class="footer"> sections used in the previous example. The corresponding example thus looks as follows (see Listing 4.13).

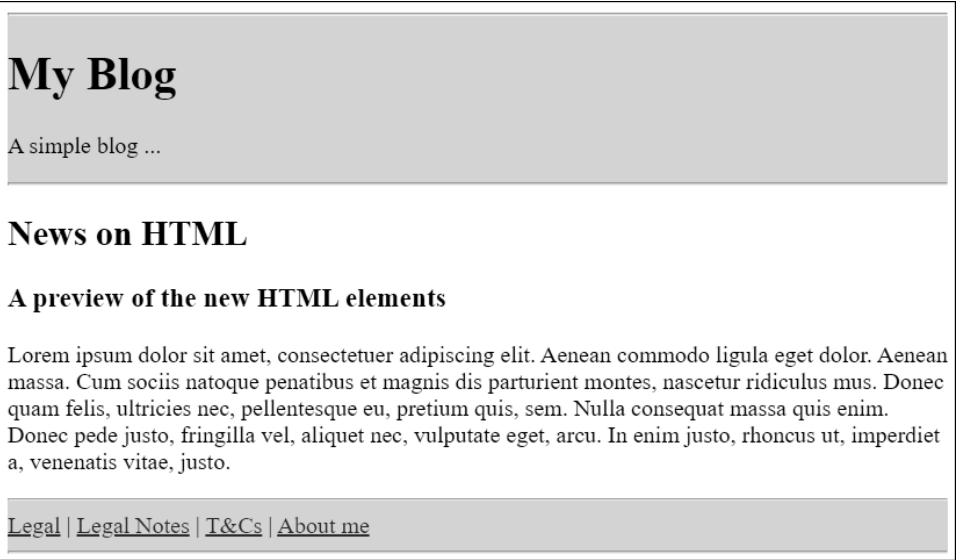


Figure 4.17 The Header and Footer of the HTML Document Appear in Gray

```
...
<body>
  <header>
    <hr />
    <h1>My Blog</h1>
    <p>A simple blog ...</p>
    <hr />
  </header>
  <h2>News on HTML</h2>
  <article>
    <h3>A preview of the new HTML elements</h3>
    <p>Lorem ipsum dolor ... </p>
  </article>
  <footer>
    <hr />
    <a href="#">Legal</a> |
    <a href="#">Legal Notes</a> |
    <a href="#">T&Cs</a> |
    <a href="#">About me</a>
  </footer>
</body>
...
```

Listing 4.13 /examples/chapter004/4_2_7/index2.html

4.2.8 Using <main>: An HTML Element for the Main Content

I described the `div` element in the previous section, so it makes sense to deal with the `main` element at this point. Where `<div id="main">...</div>` was used in the past, you can use `<main>...</main>` from now on. The `id` attribute identifies an element that occurs only once within a document.

Like all other new HTML elements, you should use the `main` element as sensibly as possible. In practice, you use it for the main content of a website, which means it's best not to place it inside `<article>`, `<aside>`, `<footer>`, `<nav>`, or header elements.

In the web browser, the `main` element is rendered like the `div` element with no special properties and only creates a line break. However, unlike the `div` element, you should use the `main` element only once (visibly) in an HTML document. In contrast to the `<section>` element, the `main` element isn't a section element, but a pure grouping element. Thus, the use of such a section doesn't affect the heading structure (the document outline) of the HTML document.

Here's an example of how you can group a section as the main section of a web page:

```
...
<body>
<header>
  <h1>My Blog</h1>
  <p>A simple blog ...</p>
</header>
<main>
  <h2>News on HTML</h2>
  <article>
    <h3>A preview of the new HTML elements</h3>
    <p>Lorem ipsum dolor...</p>
  </article>
</main>
<footer>
  <a href="#">Legal</a> |
  <a href="#">Legal Notes</a> |
  <a href="#">T&Cs</a> |
  <a href="#">About me</a>
</footer>
</body>
...
```

Listing 4.14 /examples/chapter004/4_2_8/index.html

Using <main> Multiple Times?

`<main>` is intended to present the main content of an HTML document and should therefore be included only once in a document. If it's used more than once, then this page won't pass the validation check. Nevertheless, there are *single-page* web applications, that is, applications that consist of a single HTML document and whose content is dynamically reloaded, where this rule can become an issue. For this reason, the use of the `main` element has been adjusted somewhat, and multiple `main` elements can now be used. However, only one `<main>` element of those can be visible at a time. All other `main` elements must be provided with the `hidden` attribute. For example:

```
<main>...</main>
<main hidden>...</main>
<main hidden>...</main>
```

Although there are other ways in CSS to hide individual elements, you can use only the `hidden` attribute with `<main>` for the HTML document to be valid. All other options are invalid.

4.2.9 Labeling Content Separately Using <figure> and <figcaption>

To set off or group certain content such as tables, images, listings, videos, or other HTML elements from the usual body text, you can use the `figure` element. If you want to link this section with an (optional) caption, you should use the `figcaption` element. Like the `figure` element, the `figcaption` element can contain other HTML elements besides ordinary body text. Thus, the `figure` element serves as the semantic parent for an element belonging to the page content, such as an image, table, listing, or other content, and the `figcaption` element encloses the subtitle to that element.

Here's a simple example, the result of which is shown in Figure 4.18:

```
...
<h2>HTML</h2>
<article>
  <h3>figure and figcaption in use</h3>
  <p>The text before figure ...</p>
  <figure>
    
    <figcaption>Figure 1: Once upon a time ...</figcaption>
  </figure>
  <p>The text after figure</p>
</article>
...
```

Listing 4.15 /examples/chapter004/4_2_9/index.html

If you want to place the (optional) caption with the `figcaption` element before the content (above the image in the example), you need to use the element right after the opening `<figure>`. However, it's only possible to use a `figcaption` element between `<figure>` and `</figure>`, and `<figcaption>` must be the first or last element of the figure element.



Figure 4.18 In the `<article>` Element between `<figure>` and `</figure>`, an Image Has Been Inserted with the `` Element and a Caption with the `<figcaption>` Element

Between `<figure>` and `</figure>` you can also use more than one content type (e.g., an image in the example). In the web browser, a `figure` usually doesn't get displayed separately. In addition to a separate line, the content between `<figure>` and `</figure>` is often displayed slightly indented. However, CSS is used for the design of the figure element anyway.

4.2.10 Creating Unordered Lists Using `` and ``

An unordered list is basically nothing more than an unnumbered bulleted list in which all list entries are given a bullet character. The web browsers usually display this bullet with a *bullet point*.

You can introduce such a list with an opening `` (`ul` = *unordered list*), followed by the actual bullet points, which you write between `` and ``. Each `li` element (`li` = *list item*) is a bullet point. At the end, you must end the unordered bullet list with the closing

``. Only `li` elements can be contained between `` and ``. In between the `li` elements, you can also use other HTML elements (except for section elements).

Here's a simple example of an unordered list, the execution of which you can see in Figure 4.19.

```
...
<article>
  <h2>Unordered bullet list with ul</h2>
  <ul>
    <li>Lorem ipsum dolor sit amet</li>
    <li>Donec quam felis ultricies</li>
    <li>Nulla consequat massa quis</li>
    <li>Etiam ultricies nisi vel</li>
    <li>Donec vitae sapien ut libero</li>
  </ul>
</article>
...
```

Listing 4.16 /examples/chapter004/4_2_10_15/index.html



Figure 4.19 Bulleted Lists with the `` Element Are Usually Displayed with a Bullet Point

4.2.11 Creating Ordered Lists Using `` and ``

What you've just read about the `ul` element also applies to the `ol` element (`ol` = *ordered list*). The only exception is that the `ol` element is an ordered list—more precisely, a numbered list in which the individual `li` elements are automatically numbered.

Here's an example of an ordered list, the execution of which you can see in Figure 4.20.

```
...
<article>
  <h2>Numbered bullet list with ol</h2>
  <ol>
    <li>Lorem ipsum dolor sit amet</li>
    <li>Donec quam felis ultricies</li>
    <li>Nulla consequat massa quis</li>
  </ol>
</article>
...
```

```
<li>Etiam ultricies nisi vel</li>
<li>Donec vitae sapien ut libero</li>
</ol>
</article>
...
```

Listing 4.17 /examples/chapter004/4_2_10_15/index.html



Figure 4.20 The Numbered List with the `` Element Uses Arabic Numerals by Default

4.2.12 Reversing the Numbering of an Ordered List

With HTML, it’s also possible to reverse the order of numbering via the HTML attribute `reversed` in the opening `` tag, so that the numbering gets displayed in descending order. Based on the preceding example, you only need to insert the following:

```
<ol reversed="reversed">
...
</ol>
```

Besides `<ol reversed="reversed">`, you can also just use `<ol reversed>` here because it’s a standalone attribute. But if you want to be XHTML compliant, you must use the form `<ol reversed="reversed">`. In HTML, you can use both versions.



Figure 4.21 The Numbering Order Was Reversed via the “reversed” Attribute

4.2.13 Changing the Numbering of an Ordered List

You can use the HTML attribute `start` to specify the start value of the first `li` element in the opening `` tag. All values that follow the first `li` element are incremented by the

value 1. Even within an opening `` tag, you can use the HTML attribute `value` to change the numbering of the list entry. All subsequent entries are incremented by the value 1 using the value specified in `value`.

The execution of the following example is shown in Figure 4.22.

```
...
<ol start="20">
  <li>Lorem ipsum dolor sit amet</li>
  <li>Donec quam felis ultricies</li>
  <li>Nulla consequat massa quis</li>
  <li value="101">Etiam ultricies nisi vel</li>
  <li>Donec vitae sapien ut libero</li>
</ol>
...
```

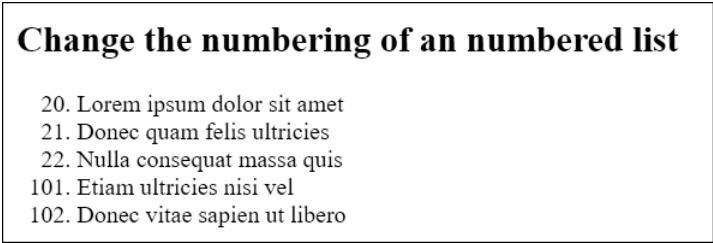


Figure 4.22 The Starting Numbering Was Set to 20 Right in the Opening `` Tag with the Attribute “start” and Then Again in an Opening `` Tag with the Attribute “value” to 101

4.2.14 Nesting Lists within Each Other

You can nest both numbered lists and bulleted lists. Such nested lists are used when, for example, you need a finer structuring of the lists, such as a table of contents (e.g., at the beginning of this book). A navigation with submenus is also often formulated by means of a bulleted list.

Nesting lists can get a little messy, so if you have a deeper bulleted hierarchy, you should use indentations and/or add a comment. When nesting, the `li` elements aren’t nested inside each other, as might be assumed, but a `ul` or `ol` element must be written again with the nested `li` elements inside an opening parent `` tag. Only when you close the opening `` tag with `` will this list be marked and displayed as a child list.

The execution of the following code snippet is shown in Figure 4.23.

```
...
<h2>Nesting bullet lists ul</h2>
<ul>
  <li>Lorem ipsum dolor sit amet
```

```
<ul><!-- Start: 1. Nesting -->
  <li>Donec quam felis ultricies</li>
  <li>Nulla consequat massa quis</li>
</ul><!-- End: 1. Nesting -->
</li>
<li>Etiam ultricies nisi vel</li>
<li>Donec vitae sapien ut libero</li>
</ul>
...
<h2>Nesting numbered lists ol</h2>
<ol>
  <li>Lorem ipsum dolor sit amet
  <ol><!-- Start: 1. Nesting -->
    <li>Donec quam felis ultricies</li>
    <li>Nulla consequat massa quis</li>
  </ol><!-- End: 1. Nesting -->
  </li>
  <li>Etiam ultricies nisi vel</li>
  <li>Donec vitae sapien ut libero</li>
</ol>
...
```

Listing 4.18 /examples/chapter004/4_2_10_15/index.html

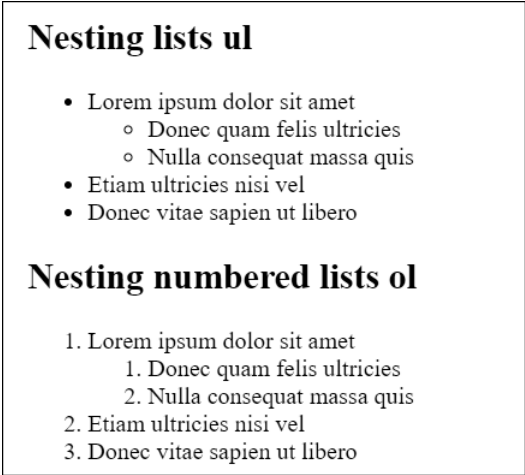


Figure 4.23 The Nesting of Unnumbered Lists and Numbered Lists during Execution

Of course, you can nest the lists even deeper. Mixing unordered and ordered lists is also possible without any problem. Unfortunately, it isn’t possible to force automatic numbering such as 1.2, 1.3, 1.4, and so on for the numbered sublists.

To illustrate this, the following is an example of a deeper and mixed nesting, the result of which is shown in Figure 4.24.

```
...
<h2>Deeper nesting and mixing lists</h2>
<ol>
  <li>Lorem ipsum dolor sit amet
  <ol><!-- Start: 1. Nesting -->
    <li>Donec quam felis ultricies</li>
    <li>Nulla consequat massa quis
    <ol><!-- Start: 2. Nesting -->
      <li>Donec quam felis ultricies</li>
      <li>Nulla consequat massa quis</li>
    </ol><!-- End: 2. Nesting -->
    </li>
  </ol><!-- End: 1. Nesting -->
  </li>
  <li>Etiam ultricies nisi vel
  <ul><!-- Start: 1. Nesting (bullet point) -->
    <li>Donec quam felis ultricies</li>
    <li>Nulla consequat massa quis</li>
  </ul><!-- End: 1. Nesting (bullet point) -->
  </li>
  <li>Donec vitae sapien ut libero</li>
</ol>
...
```

Listing 4.19 /examples/chapter004/4_2_10_15/index.html

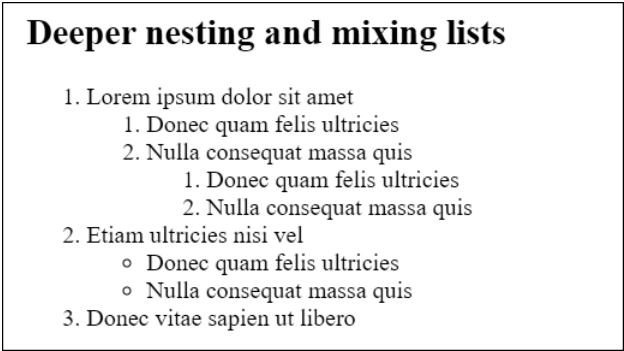


Figure 4.24 Further Nesting Depths and Mixing of Ordered and Unordered Lists

Omitting the Closing Tag from Lists

As you may remember from Chapter 2, Section 2.1.6, it’s possible to omit the closing tags in some places. Especially if a list is deeply and extensively nested, this may even be clearer and easier than setting the closing tags. As mentioned earlier, this style isn’t used in this book, and I’ve never used it in practice (yet). Nevertheless, it should be pointed out here because the lists are listed as a pro-argument, especially by the “omission faction.”

4.2.15 Creating a Description List Using <dl>, <dt>, and <dd>

In HTML, there’s another type of list you can use—the description list. This list is more of a name-value mapping list. Typical use cases for the description list are glossaries or the listing of special metadata and values; in other words, it’s simply a special list with certain data in which a value or a description is assigned.

A description list gets summarized between <dl> and </dl> (dl = *description list*). The dl element may only contain the dt and dd elements described in the same way. The expression to be described, that is, the name of the name-value mapping list, is marked with <dt> and </dt> (dt = *description term*). The associated description is written after the dt element between <dd> and </dd> (dd = *definition description*). In turn, other HTML elements may be used in dt and dd elements—except for *grouping* elements and HTML elements for new sections (*sectioning*).

Here’s a simple example of a description list, the result of which is shown in Figure 4.25. By default, web browsers display the descriptions (<dd> elements) slightly indented compared to the expression (<dt> elements). Here the description list was used for a list of abbreviations in the web jargon.

```
...
<h3>Web lingo</h3>
<dl>
  <dt>4U</dt>
  <dd>For you</dd>
  <dt>ACK</dt>
  <dd>Acknowledgment</dd>
  <dt>ASAP</dt>
  <dd>As soon as possible</dd>
  <dt>FYI</dt>
  <dd>For your information</dd>
</dl>
...
```

Listing 4.20 /examples/chapter004/4_2_10_15/index.html

Description lists with dl, dt and dd

Web lingo

4U	For you
ACK	Acknowledgment
ASAP	As soon as possible
FYI	For your information

Figure 4.25 Descriptions (<dd> Elements) Slightly Indented Compared to the Expression (<dt> Elements)

Such a name-value pair list can also be used within other HTML elements such as between <aside> and </aside> or the new details element, as shown in the following example:

```
...
<h3>Book launch</h3>
<br>
<details>
  <summary>Book information:</summary>
  <dl>
    <dt>Publisher</dt>
    <dd>Rheinwerk Verlag</dd>
    <dt>Author</dt>
    <dd>Juergen Wolf</dd>
    <dt>Scope</dt>
    <dd>400 pages</dd>
    <dt>Price</dt>
    <dd>$24.90</dd>
    <dt>ISBN</dt>
    <dd> ISBN 978-3-8362-7777-8 </dd>
  </dl>
</details>
...
```

Listing 4.21 /examples/chapter004/4_2_10_15/index.html



Figure 4.26 The Description List for an Image (a Book) Has Been Wrapped inside the <details> Element, Allowing the Description to be Expanded and Collapsed

4.3 Using Semantic HTML

Now that you know the HTML elements for page structuring and text structuring, you may be wondering how you can create a basic web page with these HTML elements so that it makes sense semantically. Specifically, this means that you can define the different logical parts of a web page with HTML tags. By the way, the semantic web isn't just a fad, but helps search engines, for example, better allocate the sheer flood of data on the internet. Search engines such as Google even prefer semantic web pages and searches HTML pages for semantic content.

Let's take as a simple example the term *goal*, whose meaning in hockey is different from that in business. The term gets its assignment and meaning only if you provide the relevant context. This is roughly how you can imagine the semantic web: you contextualize the content with code so that machines can also interpret and process it.

4.3.1 HTML without a Precise Structure

The first example is a classic HTML document that has no detailed structure:

```
...
<h1>My Blog</h1>
<p>A blog with yummy recipes ...</p>
```

```
<p>Navigation:
  <a href="#">Blog</a> | <a href="#">Recipes</a> |
  <a href="#">About me</a> | <a href="#">Legal Notes</a>
</p>
<h2>Old Posts</h2>
<ul>
  <li><a href="#">Last Week</a></li>
  <li><a href="#">Archive</a></li>
</ul>
<h2>Tasty homemade vanilla sauce</h2>
<p>Today I want to show you how ...</p>
<h3>Similar recipes</h3>
<ul>
  <li><a href="#">Chocolate sauce made from cocoa</a></li>.
  <li><a href="#">Custard Made Easy</a></li>
</ul>
<p>
  <a href="#">Contact</a> | <a href="#">FAQs</a> |
  <a href="#">About me</a> | <a href="#">Legal Notes</a>
</p>
...
```

Listing 4.22 /examples/chapter004/4_3_1/index.html

There isn't much to note about this example. The HTML code is valid and can be used like that. You can see some headings, unordered lists, navigation, and various paragraph texts. However, such code is rarely used because it's nearly unstructured and is relatively poorly suited for styling or laying out via CSS.

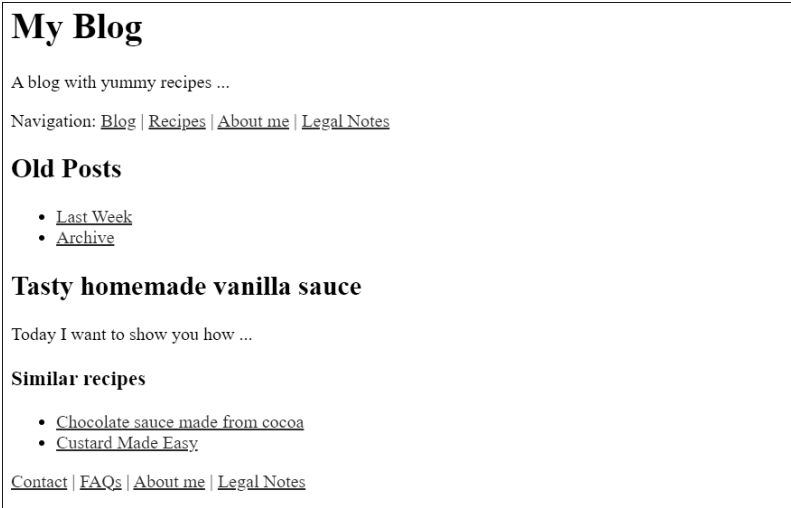


Figure 4.27 /examples/chapter004/4_3_1/index.html When Displayed in the Web Browser

4.3.2 Generic Structuring Using <div>

The first example didn't contain any element to tell you where the different sections of content were located. For this reason, we'll now use the `div` element to divide the content into separate sections. Take a look at the same example again now, but this time it contains the `div` elements:

```
...
<div>
<h1>My Blog</h1>
<p>A blog with yummy recipes ...</p>
</div>
<div>
<p>Navigation:
  <a href="#">Blog</a> |
  <a href="#">Recipes</a> |
  <a href="#">About me</a> |
  <a href="#">Legal Notes</a>
</p>
</div>
<div>
<h2>Old Posts</h2>
<ul>
  <li><a href="#">Last Week</a></li>
  <li><a href="#">Archive</a></li>
</ul>
</div>
<div>
<h2>Tasty homemade vanilla sauce</h2>
<p>Today I want to show you ... </p>
<h3>Similar recipes</h3>
<ul>
  <li><a href="#">Chocolate sauce made from cocoa</a></li>.
  <li><a href="#">Custard Made Easy</a></li>
</ul>
</div>
<div>
<p>
  <a href="#">Contact</a> |
  <a href="#">FAQs</a> |
  <a href="#">About me</a> |
  <a href="#">Legal Notes</a>
</p>
```

```
</div>
...
```

Listing 4.23 /examples/chapter004/4_3_2/index.html

This time, the content was separated by means of `div` elements. Nevertheless, we still don't see any semantic elements. Visually, nothing changes here compared to the */examples/chapter004/4_3_1/index.html* example from the previous section.

All that can be achieved by using the `div` element is to group a piece of content together. Thus, it depends on the author of the web page to assign meaning to the individual `div` elements. Before semantic elements came into play, this was done via attributes in the opening `<div>` tag. So, let's now take a look at the next step and the next example, in which the individual `div` elements get their meaning:

```
...
<div id="header">
<h1>My Blog</h1>
<p>A blog with yummy recipes ...</p>
</div>
<div id="navigation">
<p>Navigation:
  <a href="#">Blog</a> |
  <a href="#">Recipes</a> |
  <a href="#">About me</a> |
  <a href="#">Legal Notes</a>
</p>
</div>
<div id="sidebar">
<h2>Old Posts</h2>
<ul>
  <li><a href="#">Last Week</a></li>
  <li><a href="#">Archive</a></li>
</ul>
</div>
<div id="content">
<h2>Tasty homemade vanilla sauce</h2>
<p>Today I want to show you ...</p>
<h3>Similar recipes</h3>
<ul>
  <li><a href="#">Chocolate sauce made from cocoa</a></li>.
  <li><a href="#">Custard Made Easy</a></li>
</ul>
</div>
<div id="footer">
```

```
<p>
  <a href="#">Contact</a> |
  <a href="#">FAQs</a> |
  <a href="#">About me</a> |
  <a href="#">Legal Notes</a>
</p>
</div>
...
```

Listing 4.24 /examples/chapter004/4_3_2/index2.html

While nothing has changed in a purely visual sense, the `div` elements have gained meaning thanks to the `id` attribute. We now have a header, navigation, sidebar, content, and footer as it's visually represented in Figure 4.28. Using CSS, you can design and lay out these areas individually. This way, you can virtually already achieve a semantically correct structuring of the website, but not yet a semantically unified structuring.

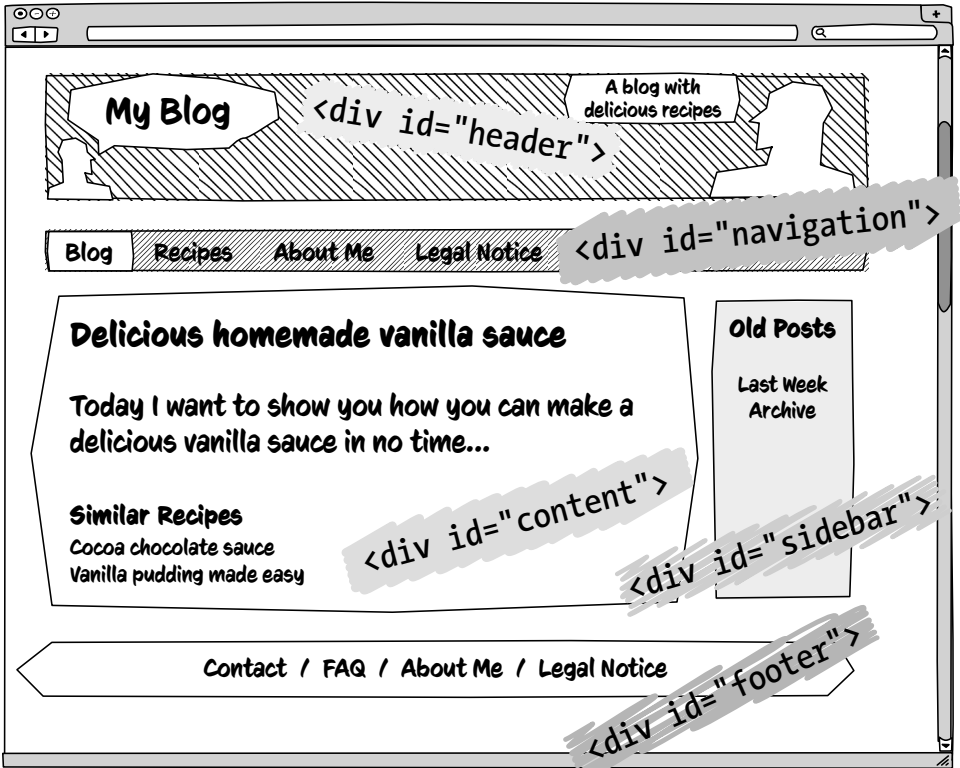


Figure 4.28 The Meaning for the Layout Areas Is Assigned via `<div>` and the “`id`” Attribute

So why should you use semantic structuring at all when you can work with `div` elements without any problem? There are several reasons for this: Despite the use of IDs

in the `div` element, you have a semantically neutral element. It isn't a standardized structuring, but instead everyone can define what they want. For the machines, there's still no difference here. They can't know what you really mean by `id="header"` or `id="content"` and what's behind it. You might as well write `id="head"` or `id="synopsis"` or whatever in all the languages of the world.

For example, imagine a smart screen reader reading the main content of the web page to a visually impaired person. How would the screen reader know what the main content is? One web developer may write `id="content"`, another may write `id="main"`, and you may write `id="musings"`. In addition, some web developers don't mark up the main content at all.

The situation is the same with search engines. For search engines to return a better result, it's helpful if they know what belongs to the main content of the web page. Again, the search engine faces nonstandard class and ID names. Thus, it's an advantage here if you tell the web crawler on the next visit: this is the main content of my site.

<div> Can Still Be Used in HTML

Using `div` elements and labeling the layout sections with the ID and class names are by no means incorrect—they represent valid HTML. In addition, the `div` element often helps you solve a problem. Nevertheless, for future projects, you should use the new semantically meaningful elements that were introduced especially for this purpose.

4.3.3 Semantic Structuring Using the Elements Provided in HTML

To write a semantically meaningful structure as HTML code for machines, there are suitable elements in HTML that have already been described in the book and are listed once again in Table 4.3.

HTML Element	Meaning	Section in This Chapter
header	Header sections	Section 4.1.4
nav	Navigation blocks	Section 4.1.2 and “Declaring Content as a Page Navigation Bar Using <code><nav></code> ” on Page 102
section	Division into content sections	“Dividing Content into Topic-Based Sections Using <code><section></code> ” on Page 97
article	Division into self-contained blocks	“Dividing Content into a Self-Contained Block Using <code><article></code> ” on Page 98
aside	Additional Information	Section 4.1.2
footer	Footer sections	Section 4.1.4

Table 4.3 Semantic HTML Elements

Returning to our example `/examples/chapter004/4_3_2/index2.html`, this HTML code should do without `div` elements and instead rely on semantic HTML elements:

```
...
<header>
<h1>My Blog</h1>
<p>A blog with yummy recipes ...</p>
</header>
<nav>
<p>Navigation:
  <a href="#">Blog</a> |
  <a href="#">Recipes</a> |
  <a href="#">About me</a> |
  <a href="#">Legal Notes</a>
</p>
</nav>
<aside>
<h2>Old Posts</h2>
<ul>
  <li><a href="#">Last Week</a></li>
  <li><a href="#">Archive</a></li>
</ul>
</aside>
<article>
<h2>Tasty homemade vanilla sauce</h2>
<p>Today I want to show you ...</p>
<h3>Similar recipes</h3>
<ul>
  <li><a href="#">Chocolate sauce made from cocoa</a></li>.
  <li><a href="#">Custard Made Easy</a></li>
</ul>
</article>
<footer>
<p>
  <a href="#">Contact</a> |
  <a href="#">FAQs</a> |
  <a href="#">About me</a> |
  <a href="#">Legal Notes</a>
</p>
</footer>
...
```

Listing 4.25 `/examples/chapter004/4_3_3/index.html`

When you look at the HTML document with the semantic elements, it's probably already much easier to recognize at first glance what has which meaning here. This is just a simple example. Here you can immediately see the header, navigation, sidebar, main content, and footer (see Figure 4.29). This way, the content could perhaps also be placed inside the `main` element in which the individual articles are then summarized using the `article` element.

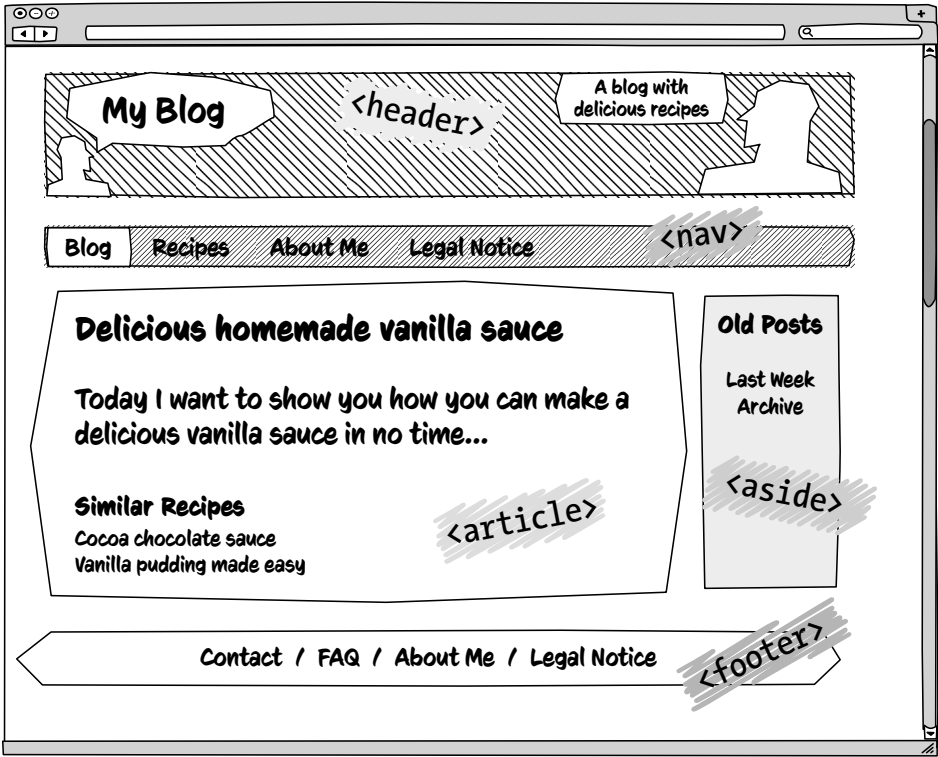


Figure 4.29 Layout Areas Marked with HTML Semantic Elements

The point here isn't at all where you can use exactly which HTML element in detail, but rather that this semantic structuring makes sense, even if you've never heard of new elements such as `nav`, `article`, `header`, `footer`, and so on. The logic here is almost self-evident. It's much easier to see where the navigation, header, or footer is written in this document.

The Main Content with `<main>`

If you still want to summarize the main content of the web page in `<main>` in the example `/examples/chapter004/4_3_3/index.html`, then you should choose the `article` element.

4.3.4 What’s the Use of Those Semantic HTML Elements?

If you’ve carefully followed this section, you’ve seen that the semantic HTML elements are very useful. Thus, probably one of their advantages is that they make life easier for you as the developer of the website.

For “normal” visitors, these semantic HTML elements don’t have much value at first. On the contrary, those visitors won’t even be able to distinguish in the web browser whether you’ve used `div` elements or the semantic HTML elements. However, if, for example, a new smart web browser provides special features that let you get to navigation by clicking a button, the new semantics take on meaning for normal visitors as well.

The situation is different, however, for visually impaired visitors who use a screen reader. A good screen reader could “recognize” the content of the web page based on the new semantic structure and thus jump directly to the content or navigation.

Of course, you shouldn’t disregard the search engines at all. For example, you could let the search engine know in a consistent and standardized way where which content is located, so that it assigns a higher ranking to the relevant content of a web page.

4.4 HTML Elements for Text Markups

You apply HTML elements for text markup within plain text for individual letters, words, or parts of sentences. Thus, the described elements don’t create a new paragraph or line break, but mark out specific passages in a continuous text according to the semantics defined for the element. You can find all text markups used here in the HTML document `/examples/chapter004/4_4/index.html`.

Text Formatting via CSS

Even though many of the elements presented here cause a slight visual change of the text in the web browser, you shouldn’t use these HTML elements for text formatting. CSS is responsible for text formatting. These HTML elements rather serve a clean semantic text markup. Thanks to semantic text markup, you can lay the foundation for later text formatting with CSS. If you use sensible text markup in body text, you can later format your text more easily and logically with CSS.

HTML Element	Meaning
<abbr>	Marking abbreviations or acronyms.
<cite>	Marking text as source text of a working title.

Table 4.4 Brief Overview of the Elements Covered for Text Markups

HTML Element	Meaning
<code>	Marking up computer code within a paragraph of text.
<pre>	Marking up preformatted text. All spaces and line breaks get displayed as specified in the text.
<kbd>	Marking up text as keyboard input.
<samp>	Marking up text as screen output of a program.
<dfn>	Defining a term.
<var>	Marking up text as a variable.
<bdo>	Changing the text direction for bidirectional text.
<bdi>	Defining a section for bidirectional text.
	Highlighting text you would emphasize in spoken language.
	Highlighting words or passages that are particularly important in terms of content.
<i>	Marking up words or passages with technical terms, thoughts, and foreign words.
	Marking up meaningful names or keywords.
<mark>	Highlighting text with a marker.
<q>	Marking up words or passages as cited or spoken text.
<u>	Marking up text underlined as proper name or incorrect words or passages.
<s>	Marking up text as no longer valid or obsolete.
<ins>	Marking up text as newly added in the revised sense.
	Marking up text as deleted in the revised sense.
<sub>	Marking text as subscript.
<sup>	Marking text as superscript.
<time>	Marking up dates and times
<small>	Marking up text as small print, such as for copyright information, licensing information, or legal notes.
<ruby>, <rp>, and <rt>	Specifying Ruby annotations.
	Marking up a general section within a paragraph of text.

Table 4.4 Brief Overview of the Elements Covered for Text Markups (Cont.)

4.4.1 Marking Up Abbreviations or Acronyms Using <abbr>

The `abbr` element (`abbr` = *abbreviation*) can be used for abbreviations or acronyms. It’s also helpful to use the global HTML attribute `title` in which you write out the abbreviation or acronym so that a web browser can display the full meaning when hovering over it, as you can see in Figure 4.30. This reproduces the code snippet of the following example:

```
...
<p>The <abbr title="world wide web">WWW</abbr> is teeming with
  abbreviations.
</p>
...
```

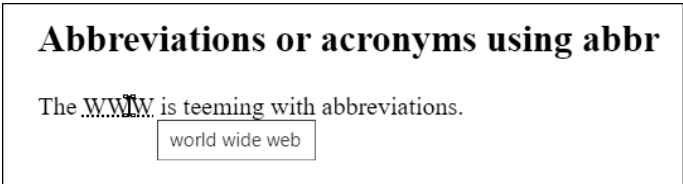


Figure 4.30 The Global “title” Attribute Displays the Meaning of the Abbreviation “WWW” When You Hover the Mouse Cursor over the Word

Anyone who writes abbreviations between `<abbr>` and `</abbr>` is passing useful information to the web browser, language-checking software, translation systems, screen readers, or even the search engines for indexing. The extent to which this information is useful and actually used can’t always be predicted. Nevertheless, the `abbr` element is very useful for logical text markup.

4.4.2 Marking Up Text as the Source of a Working Title Using <cite>

You can use the `cite` element when you include the title of a book, movie, painting, piece of music, exhibition, and so on in the body text. However, you should only mark up the working title and not the name or main character of the title. Again, you can usually still use the global HTML attribute `title` to specify more information about the working title when the user hovers over it with the mouse pointer. Most web browsers display everything between `<cite>` and `</cite>` in italics.

```
...
<p>According to the book <cite>HTML and CSS—The Comprehensive
  Handbook</cite> it should read:
</p>
...
```

According to the book *HTML and CSS—The Comprehensive Handbook* it should read:

In HTML 4.01, this element was also used for short quotations. In new HTML, the semantic meaning has been converted and should now only be used for working titles.

Figure 4.31 In This Example, We Wrote the Working Title of a Book between <cite> and </cite>

4.4.3 Marking Up Computer Code Representation Using <code> and <pre>

You should use the `code` element to indicate computer code within body text. Most web browsers often display this area using a monospace font such as Courier, as shown in Figure 4.32, which renders the following code snippet in the web browser:

```
...
<h2>Computer code with <code>code</code></h2>
<p>The <code>code</code> element does not contain any attributes.</p>
...
```



Figure 4.32 The <code> Element Is Suitable for Marking Up Language Elements or Parts of a Source Code of a Particular Language

If you want to format multiple lines of computer code, you should note the `code` elements in between `<pre>` and `</pre>`. The `pre` element (`pre` = *preformatted*) represents preformatted text. In the section between `<pre>` and `</pre>`, several whitespace characters won’t get combined to one space, but everything is output as it was entered in the editor. Because that section is output in a monospace font, the `pre` element is very suitable for outputting source text across multiple lines. This isn’t to say that `<pre>` is only suitable for marking up source code. It’s therefore recommended that you specify the content between `<pre>` and `</pre>` more precisely with appropriate text markup. So, for source code, you should use `<code>`; for keyboard input, `<kbd>`; and for displaying a program output, `<samp>`.

In the following example, the text preformatted between `<pre>` and `</pre>` is output as it was written. Specifying `<code>` and `</code>` between `<pre>` and `</pre>` isn’t mandatory, but it makes the text markup even more precise. You can see the following example at execution in Figure 4.33:

```
...
<p>Here is a source code snippet of a C program:</p>
<pre><code>#include <stdio.h>

int main(void)
{
    puts("Hello World!");
    return 0;
}</code></pre>
...
```

Here is a source code snippet of a C program:

```
#include <stdio.h>

int main(void)
{
    puts("Hello World!");
    return 0;
}
```

Listing 1: Hello World in C

Figure 4.33 The Text Preformatted between `<pre>` and `</pre>` Gets Output Exactly as It Was Entered

The Masking Characters “<” and “>”

To display `<` or `>` characters in HTML that aren’t to be used as HTML, the character entities `<` for `<` and `>` for `>` were used here.

4.4.4 Keyboard Input Using `<kbd>` and Program Output Using `<samp>`

The `kbd` element (`kbd` = *keyboard*) should be used to mark up continuous text as keyboard input. The `samp` element, on the other hand, should be used for the screen output of programs. Most often, these two elements are also rendered in a monospace font (usually Courier) in the web browser, as you can see in Figure 4.34, which shows the following example running in the web browser:

```
...
<p>You can use <kbd>Strg</kbd> + <kbd>A</kbd> to mark up the entire text.</p>
<pre>term#1&gt; <kbd>gcc -o Wall hello hello.c</kbd>
term#1&gt; <kbd>./hello</kbd>
<samp>Hello World!</samp>
term#1&gt;</pre>
...
```

You can use **Strg + A** to mark up the entire text.

```
term#1> gcc -o Wall hello hello.c
term#1> ./hello
Hello World!
term#1>
```

Figure 4.34 The Web Browsers Themselves Decide How to Display the Text Between `<kbd>` and `</kbd>` for Input or `<samp>` and `</samp>`

In Figure 4.34, the `kbd` elements have been made bold, and the `samp` elements have been made gray by using CSS to help you see what has been used where.

4.4.5 Marking Up Text as a Definition Using `<dfn>`

Text that you write between `<dfn>` and `</dfn>` is supposed to represent a definition. Usually, you mark up a word or a text passage, which you then explain in the text that follows. However, the `dfn` element shouldn’t mark up the definition itself, but the defined term. Let’s take a look at a simple example:

```
...
<p>A <dfn>smartphone</dfn>—as opposed to a
    cell phone—provides more
    computer functionality and better connectivity.
</p>
...
```

A *smartphone*—as opposed to a cell phone—provides more computer functionality and better connectivity.

Figure 4.35 In this Paragraph Text, the Term “Smartphone” Was Described, Which Is Why It Was Placed between `<dfn>` and `</dfn>`

In common practice, you can also use another element such as `<abbr>` inside `<dfn>` and `</dfn>`, as shown in the following example:

```
...
<p>A <dfn><abbr>smartphone</abbr></dfn>—as opposed to
    a cell phone—provides more
    computer functionality and better connectivity.
</p>
...
```

You can also use the global attribute `title` inside the opening `<dfn>` tag. The value of `title` should be the same as the content of the `dfn` element.

4.4.6 Marking Up Text as a Variable Using <var>

You can use the `var` element to mark up the text in between as a variable. Such a variable can be, for example, part of an application, a mathematical expression, or an identifier of a variable in a programming language:

```
...
<p>The radius <var>r</var> is equal to
    half the diameter <var>d</var>.
</p>
...
```

4.4.7 Changing the Text Direction Using <bdo> and <bdi>

The `bdo` element (`bdo` = *bidirectional override*) allows you to change the text direction. This is useful, for example, when you want to display text that is written from right to left (e.g., Hebrew or Arabic). By default, the text is displayed from left to right. To change the text direction, you must use the global HTML attribute `dir`. The attribute value `rtl` makes the text run from right to left, whereas `ltr` makes it run from left to right.

You don't need to put every Hebrew or Arabic word between `<bdo dir="rtl">` and `</bdo>`. When you use Unicode in HTML, the text direction is usually automatically taken into account according to the language, provided you use a Unicode-capable web browser. You should only use the `bdo` element if the correct text direction doesn't work.

To illustrate this, here's a code snippet in which in the first paragraph text—a palindrome for fun—was put between `<bdo>` and `</bdo>`, and the text alignment was changed via the attribute `dir` into the value `rtl` (*right to left*). The second paragraph, on the other hand, displays the Hebrew word “shalom”, which usually doesn't require changing the text direction. You can see the result of these lines in Figure 4.36:

```
<p><bdo dir="rtl">Never odd or even</bdo></p>
<p>שלום</p>
```

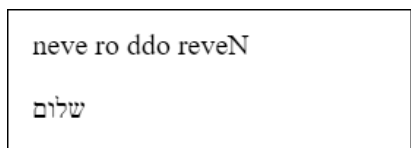


Figure 4.36 Example Executed with <bdo>

The situation is different in the following HTML lines:

```
<p>1: السلام عليكم(as-salaam alaykum)</p>
<p>2: שלום(shalom)</p>
<p>Howdy: 3</p>
```

The first two examples would probably not lead to the desired result here. Although *as-salaam alaykum* in Arabic and *shalom* in Hebrew are correctly written from right to left, the following colon and number have been given a different writing direction. Originally, this was supposed to look like the third paragraph with *Howdy*.

Here, the writing behind the Arabic characters continued from right to left, so that the colon and the number behind it also retained the writing direction. Only the translation of the Arabic or Hebrew meaning was reproduced in the correct place.

To be on the safe side, the `bdi` element (`bdi` = *bidirectional isolation*) was introduced for this purpose. Using the `bdi` element, you can mark up the boundaries of text direction changes in a Unicode-enabled web browser more accurately. Thus, in the preceding example, you only need to put the Arabic or Hebrew characters between `<bdi>` and `</bdi>`.

After that, it looks as shown in Figure 4.37:

```
<p><bdi>السلام عليكم</bdi>: 1 (as-salaam alaykum)</p>
<p><bdi>שלום</bdi>: 2 (shalom)</p>
<p>Howdy: 3</p>
```

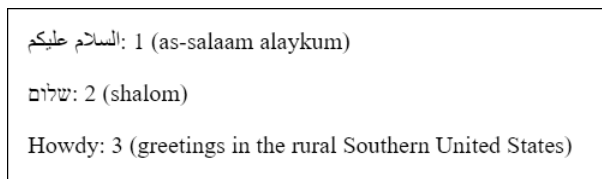


Figure 4.37 Thanks to the Containment of the Arabic and Hebrew Script between <bdi> and </bdi>, the Colon and the Decimal Number Now Display after the Script

4.4.8 Emphasizing Text Using , , <i>, and

To emphasize text, you can use either the `em` element (`em` = *emphasis*) or the `strong` element. The `em` element should be used for words or passages that you would emphasize when speaking.

If you want to bring a word or passage more into focus, you should use the `strong` element. In contrast to the `em` element, the `strong` element is used to mark certain places in the text with a special signal or warning effect. The `strong` element should definitely be used for words or passages that are particularly important in terms of content.

Let's take a look at the following example:

```
...
<p><em>Bear</em>! Who the hell is this <em>Bear</em>!</p>
<p><strong>Caution!</strong> <em>Bear</em> could be standing behind you!</p>
<p><strong>Delivery date in <strong><em>summer 2022</em></strong></strong></p>
...
```

In this example, it's semantically clear from the emphasis on *bear* in the first paragraph in Figure 4.38 that it isn't the animal that is referred to here, but a person with the surname "Bear". In the second paragraph, the word *Caution!* was marked with a special importance. In the last paragraph, an `em` element was nested within a `strong` element to emphasize *summer 2022* more strongly in terms of content in addition to its particular importance.

Bear! Who the hell is this *Bear*!

Caution! *Bear* could be standing behind you!

Delivery date in *summer 2022*

Figure 4.38 Different Ways to Emphasize or Highlight a Text Using `` and ``

Here, you could also still nest the same HTML elements to increase the emphasis or importance of `em` or `strong` elements, at least semantically.

Because web browsers usually render `` with italic and `` with bold font, you shouldn't make the mistake of replacing these elements with `<i>` and ``, respectively, because these elements (i.e., `` and `<i>` or `` and ``) are rendered quite similarly in the web browser. As mentioned at the beginning, HTML isn't used to format the text, but these HTML elements are about giving the text a meaningful significance.

The `i` element is recommended to mark special technical terms, a thought, scientific names, or foreign language words. The `b` element, on the other hand, should be used for meaningful names or keywords to draw attention to something.

Using `<i>` and ``

Standard HTML recommends using the `b` or `i` elements only if no other HTML tag fits to mark up the text or passage. The days when these elements were used purely for formatting are over because CSS is used for that purpose.

4.4.9 Highlighting Text Using `<mark>`

You should use the `mark` element to mark up words or passages in a continuous text. The easiest way to compare such a markup is with a highlighter. In practice, this HTML element should be suitable for visually highlighting the search term found during a search. This works only if the content is generated dynamically. The element is also very suitable for highlighting code fragments of a source code.

The following code snippet demonstrates the `mark` element whose execution is shown in Figure 4.39:

```
...
<p>In 2021, profits have increased by
  <mark>100 percent</mark>.
</p>
<p>Here is a source code snippet of a C program:</p>
<pre><code>#include <stdio.h>

int main(void)
{
  <mark>puts("Hello world!");</mark>
  return 0;
}</code></pre>
...
```

In 2021, profits have increased by 100 percent.

Here is a source code snippet of a C program:

```
#include <stdio.h>

int main(void)
{
  puts("Hello world!");
  return 0;
}
```

Figure 4.39 Web Browsers That Recognize the New Element Usually Mark the Text Placed between `<mark>` and `</mark>` with Yellow Background Color

Even though the `mark` element predominantly applies yellow background color to the text, you shouldn't use it to apply a background color to a text. You should rather use `<mark>` only if it makes sense in terms of content and no other semantic HTML element is suitable. If you want to format the text background, you should use the `span` element (Section 4.4.17) with CSS instead.

4.4.10 Placing Text between Quotes Using `<q>`

While you can use the `blockquote` element to quote an entire paragraph text, the `q` element allows you to quote something in the middle of the text or mark it as spoken text. Text or passages you insert between `<q>` and `</q>` should be placed between quotes by the web browser. You shouldn't use the `q` element if you simply want to enclose a word or passage in quotation marks. That wouldn't be the semantic sense of the `q` element.

If you use a quote or spoken text from another source, you can use the `cite` attribute with a URL to the source. Because the implementation of the `cite` attribute is still poor in web browsers, you may want to consider using a hypertext link.

You can also nest the `q` element. Such nested `q` elements usually get another matching quotation mark. In this country, for example, the outer quotation marks are double and the inner ones are single. This is demonstrated in the following example, the execution of which you can see in Figure 4.40:

```
...
<p>Wolf asked: <q cite="http://tom-bear.com/">
  <em>Bear</em>! Who the hell is this <em>Bear</em>!</q>
</p>
<p>Fox replied: <q>Caution! <q><em>Bear</em></q> could be
  standing behind you!</q>
</p>
...
```

As you can see in Figure 4.40, this example quoted spoken text in the first paragraph. The second paragraph demonstrates the nesting of `q` elements. The inner `q` element was placed between single quotes and the outer one between double quotes.

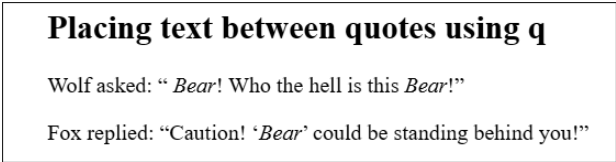


Figure 4.40 Placing Text between Quotes Using the `<q>` Element

For Advanced Users: Setting Custom Quotation Marks

The problem with the `q` element is that the quotes set by the web browser may not always be the ones you want. In this case, an intervention with CSS is a good idea. Thus, the following CSS line could be used to change the first and second nested characters of the `q` element:

```
q {quotes: '»' '«' '»' '«';}
```

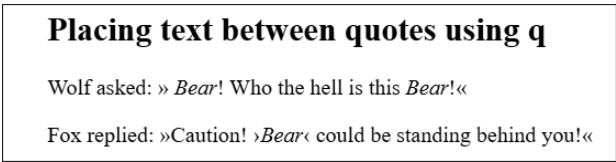


Figure 4.41 The Quotes of the `<q>` Element Have Been Changed with CSS

4.4.11 Underlining or Crossing Out Text Using `<u>` and `<s>`

Both the `u` element (`u` = *underline*) and the `s` element (`s` = *strikethrough*) were marked as deprecated with HTML 4.01 and were supposed to be removed from the standard. With

the new HTML standard, they have acquired a new semantic meaning and are thus again an official part of HTML.

You should use the `s` element to mark content as obsolete or no longer correct. The web browsers display the text between `<s>` and `</s>` as strikethrough. If you want to display a document edit where you want to mark a word or passage from the not yet finished document as deleted, you should use the `del` element instead.

According to its new meaning, the `u` element is to be used for underlining proper names, as is common, for example, in Chinese writing (see http://en.wikipedia.org/wiki/Proper_name_mark). Most readers are unlikely to use Chinese proper names, so another recommended example of the `u` element is to knowingly indicate misspelled words or passages containing errors. Web browsers usually display the `u` element with an underscore. In addition to the `u` element, there's the `ins` element, which is also rendered as underlined but is intended to indicate newly inserted content (Section 4.4.12).

Here's a short example, the execution of which you can see in Figure 4.42:

```
...
<p>You can place a text in the middle with
  <s><code>&lt;center&gt;</code> or</s> the
  CSS feature <code>text-align</code> and the value
  <var>center</var>.
</p>
<p>我来自
  <u>德国
</u>。 = I am from Germany.</p>
<p>Also, <u class="spell-checker">spellig errors</u>
  can be marked with it.
</p>
...>
```

As you can see in Figure 4.42, in the first paragraph text, the content `<center>` or was crossed out. In the second paragraph text, a Chinese proper name (Germany = 德国) is underlined. Finally, in the final paragraph, a spelling error was underlined with a red dashed line. The color changes and the style of the underline were adjusted with CSS in the example.

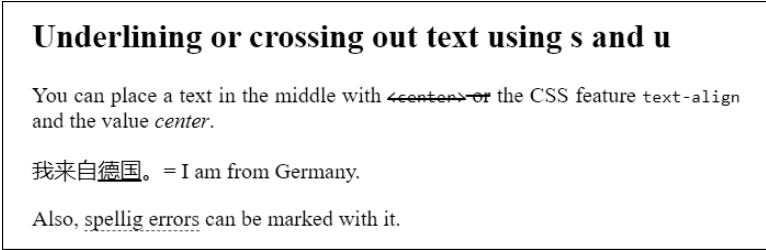


Figure 4.42 Underlining or Crossing Out Text Using `<u>` and `<s>`

In any case, it's important that you use both elements for their intended content and not for text decoration. If you want to underline or strike through text for purely decorative reasons, you should use CSS to do so.

4.4.12 Marking Changes of Text Using <ins> and

The `ins` element is rendered by web browsers similar to the `u` element, while the `del` element is rendered similar to the `s` element. Nevertheless, the semantic meaning of the two elements is different and therefore not interchangeable.

The `del` element (`del = delete`) allows you to mark a content-related (active) editing of a text as deleted in the revised sense. It's used to inform readers that this part has been revised or further developed. Instead of removing the text completely, you want to make sure that the previous versions of the text remain visible. The web browser usually crosses out this text.

The `ins` element (`ins = insert`) is the counterpart of the `del` element and should be used when something new gets inserted into the document. Here again, you mark a further development of the previous version of the text. The web browsers usually display this text with an underscore.

You can see the execution of the following example with the elements `del` and `ins` in Figure 4.43.

```
...
<del>
  <p>The singer performs on 1/1/2024 in the concert hall!</p>
</del>
<ins>
  <p>The concert was canceled,
    because the singer is sick!</p>
</ins>
...
```

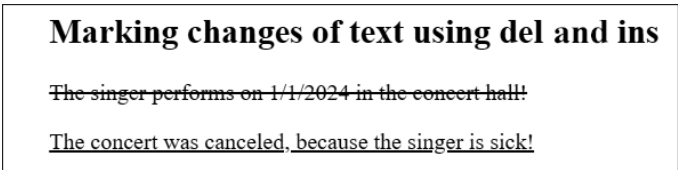


Figure 4.43 The Element Used to Delete a Paragraph Text and Insert a New Paragraph with a New Message between <ins> and </ins>

Again, you should use the `del` and `ins` elements only if they fit the semantics. If you want to cross out or underline the text in a purely decorative way, CSS should be the first choice.

Text Underline Can Be Confusing!

The frequent use of underscores with `<ins>` or `<u>` may confuse the user because hypertext links with `<a>` are usually also represented with an underscore. It's therefore also recommended to change the formatting with CSS, so that the individual elements are displayed in a clearly distinguishable manner.

4.4.13 Displaying Text as Superscript or Subscript Using <sup> and <sub>

These two markups are mainly used for simple mathematical and chemical formulas to lower text with the `sub` element (`sub = subscript`) and to raise it with the `sup` element (`sup = superscript`).

Here's a code snippet as an example, the execution of which you can see in Figure 4.44:

```
...
<p><sup>[1]</sup> Reaction scheme: 2 H<sub>2</sub><sup>0</sup>
  &RightArrow; 2 H<sub>2</sub><sup>+ 0</sup><sub>2</sub></sup></p>
<p><sup>[2]</sup><sup></sup> Calculate circular area: A = &pi; * r
  <sup>2</sup></sup></p>
...
```

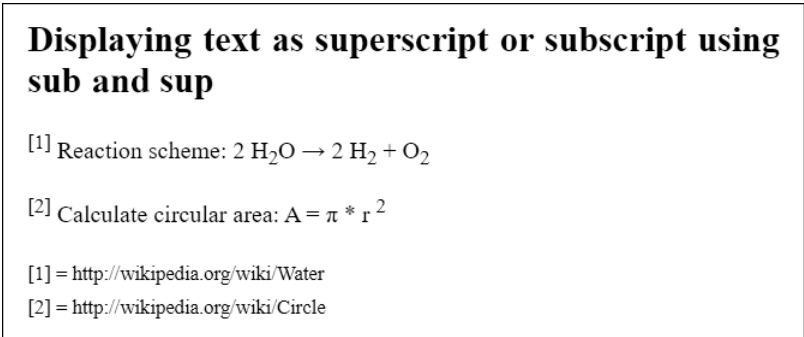


Figure 4.44 The <sub> and <sup> Elements Were Used Several Times for Superscript and Subscript Numbers and Footnotes, Respectively

4.4.14 Marking Dates and Times Using <time>

The `time` element was introduced to mark up dates and times. When displayed in a web browser, text placed between `<time>` and `</time>` is usually not visually noticeable at all. The goal and purpose of the `time` element is rather that date and time are uniquely coded for machines and can also be displayed in a readable way for humans. You can specify the machine-readable form in the HTML attribute `datetime`, while the human-readable form is usually written between `<time>` and `</time>`. Here's a brief example:

```
...
<p>We met on my <time datetime="2023-11-12">
  44th birthday</time> at <em>Bear's place</em>.
  <time datetime="20:00">at 8 pm.</time>
</p>
...
```

The specification 44th birthday can be any other text such as *Wednesday* or *November 12* as long as the value of `datetime` is a precise date of the Gregorian calendar. The same applies to the time at 8 pm.

Valid Machine-Readable Date and Time Information

A date readable by machines is specified as YYYY-MM-DD. YYYY is the year (e.g., 2023), MM is the month (e.g., 11 for November), and DD is the day in the month (e.g., 12). If you also want to note the time, you must place a capital T between the date and the time and then enter the time behind it in the form HH:MM where HH stands for hours and MM for minutes. In a newer version of time, this T can be omitted and a space can be used instead. Optionally, you can specify the time zone offset from UTC (*Coordinated Universal Time*). UTC is a designation for Universal Time. Here, you must specify a + followed by HH:MM, for example:

2023-10-10T21:00+01:00

For this example, you enter October 10, 2023, as the date. The time is exactly 9 pm, and the +01:00 at the end means UTC + 1 hour. Many other different forms of presentation are possible in this context. For more information, you should visit www.w3.org/TR/2011/WD-html5-20110525/text-level-semantics.html#the-time-element. For further and future developments of the `time` element, you may find the following website useful: http://wiki.whatwg.org/wiki/Time_element.

Alternatively, you can specify the date, time, and time zone (if needed) in `datetime` at once:

```
...
<p>We met on my
  <time datetime="2023-11-12T20:00+00:00">44th birthday
  </time> at <em>Bear's place</em> at 8pm.
</p>
...
```

So, if you use a valid `datetime` attribute with the `time` element, you can write whatever you want between `<time>` and `</time>`. *Without* specifying the `datetime` attribute, you *must* specify a valid date format and/or a valid time format—that is, the machine-readable version—between `<time>` and `</time>`, such as the following:

```
...
<p>We met on <time>2023-11-12</time> at <em>Bear's place</em>
  at <time>20:00</time>
</p>
...
```

The `time` element has been improved and made much more flexible over time after its first release. For example, the following specifications are also possible:

```
...
<p>On every <time datetime="11-12">birthday</time>
  I got flowers.
</p>
...
```

This refers to November 12. Another option would be to specify the following for date-time if you don't remember the exact day of the date:

```
<p>The concert in the photo was recorded sometime in
  <time datetime="2023-08">August</time>
  this year.
</p>
```

Here, a date in August 2023 is meant. It's possible to use only the year (e.g., `datetime="2023"`).

Another improvement is that you can use a time duration. Here's an example of how such a duration is represented:

```
<p>the rock festival lasted <time datetime="P3D">3
  days</time>.
</p>
```

The letter P stands for *period*, the D for *day*, and the 3 for three days. You can still specify a time period from a combination with H for *hours*, M for *minutes*, and S for *seconds* (e.g., `datetime="P1D5H10M"` = 1 day, 5 hours, and 10 minutes).

Here's What Doesn't Work (Yet)!

It isn't yet possible to specify a time before Christ (BC), neither can you specify a time period based on two date ranges. For this purpose, you still need to use two `time` elements.

What Should I Use `<time>` for in Practice?

If you use the `time` element, it will be easier for other programs to index this data. For example, it's easier for a script in blog articles to extract the date using the `time` element, rather than using any other techniques to look for and read this data. By having the date and/or time information in a machine-readable form, there's the advantage for search engines to make use of it when searching for items of a certain period or date.

If you then also use the `datetime` attribute, you can provide readers with a reader-friendly alternate display between `<time>` and `</time>`. Basically, you should put a (readable) date and/or time between `<time>` and `</time>` because the web browser won't display an automatic value in between if you don't put anything there.

Another possibility is that future web browsers provide the option to enter a date into the calendar at the request of the visitor. In addition, the web browser could convert the time used into the visitor's time zone if the appropriate value was specified with `datetime`. And he could convert the season according to the Buddhist time calculation, which is valid, for example, in Thailand or Laos.

In Figure 4.45, you can see again all `time` elements described and used here. I underlined the places where `<time>` was used with a dotted line using CSS for better visibility. Here, the date on which the article was written was additionally indicated behind the article title. To use the `time` element in a semantically correct way, the specification of an exact time must be used and observed. For example, you should avoid the following incorrect usage:

These are the results from `<time>last week</time>`.

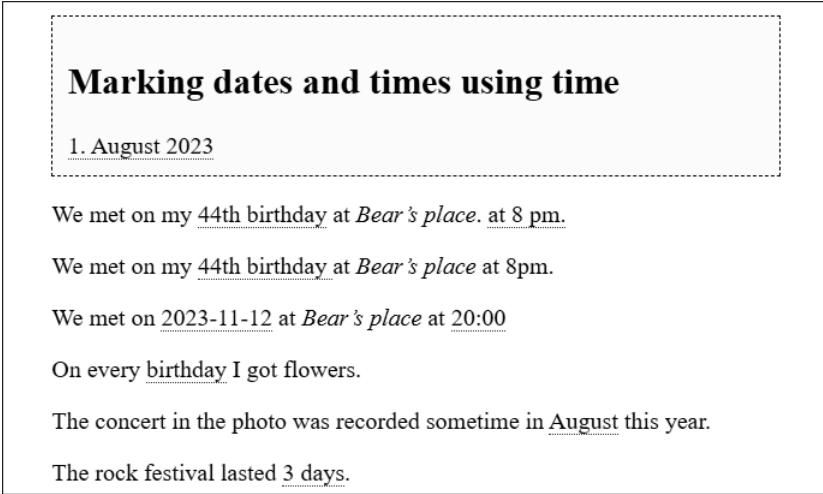


Figure 4.45 To Clarify What Is between `<time>` and `</time>`, a Dotted Underline Has Been Added

4.4.15 Marking the Small Print Using `<small>`

You should use the `small` element for words or text passages in which you want to display some small print. This can be copyright information, license information, legal notes, and so on.

Here's a short code snippet with the `small` element; its execution is shown in Figure 4.46.

```
...
<article>
  <header>
    <h2>Small print with small</h2>
    <small>&copy; John Doe;
    <time datetime="2024-01-01">January 1, 2024</time></small>
  </header>
  <p>The shipment can be delivered in <time datetime="P2D">2 days</time>.
    <small>(Due to high demand
    it can also take longer (+1 day)).</small>
  </p>
</article>
...
```

Again, you should use this element only if it semantically fits the content and not to make text look smaller. For visual adjustments, you should use CSS.



Figure 4.46 A Copyright Was Placed in the Head of an Article as well as Small Printed Information between `<small>` and `</small>`

4.4.16 Using `<ruby>`, `<rp>`, and `<rt>` for Annotations about Pronunciation

The Ruby annotation is probably of little interest to most readers. This is an annotation system in which the text appears with the annotation in one line, as used in Japanese and Chinese texts. If you want to know more about this notation, see https://en.wikipedia.org/wiki/Ruby_character.

Here, we'll only briefly describe the use of the Ruby annotation with the existing HTML elements. For this purpose, here's a simple example in which Asian characters have been omitted:


```
...
<p>
  <ruby>
    LOL<rp> (</rp><rt>Laugh Out Loud</rt><rp></rp>
  </ruby>
</p>
...
```

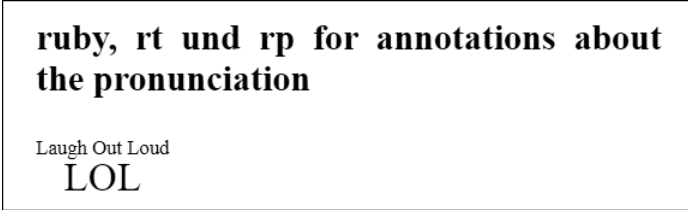


Figure 4.47 The Text between a Ruby Annotation Is Displayed as Text with Annotation in One Line

The example features LOL with annotations (which is nonsense here, of course). All characters, including annotations, are written between `<ruby>` and `</ruby>`. Then the character is noted as element content (here: LOL). The parentheses of the annotations are created with the (optional) `rp` element (`rp` = *Ruby parenthesis*). The text is then marked with the `rt` element (`rt` = *Ruby text*). Thus, the annotation is written between `<rt>` and `</rt>`.

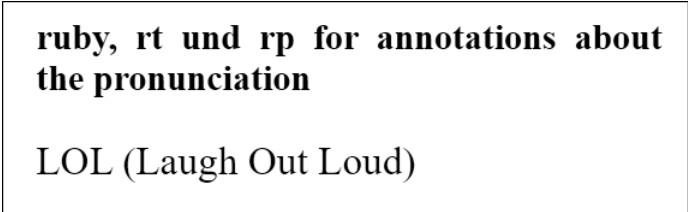


Figure 4.48 The Optional `<rp>` Element Is Used to Put Parentheses around the Ruby Text (with the `<rt>` Element) for Web Browsers That Don't Understand `<ruby>`

Web Browser Support for `<ruby>`

Web browser support in current web browsers is now quite good. In some web browsers, however, an add-on may need to be installed.

4.4.17 Grouping Ranges of Individual Text Passages Using ``

While you can use the `div` element (Section 4.2.7) to group entire groups into one block, you can use the `span` element to mark up individual passages of text inside the body

text with CSS. Visually, text placed between `` and `` doesn't change at all. In addition, the `span` element in conjunction with a JavaScript is quite useful when searching for global attributes used within it to directly update the element content. Here's a rather theoretical example:

```
<p>Current temperature <span id="temp">64</span> F</p>
```

Here, you could use a JavaScript to read the global `id` with the value `temp` and further process or update the element content. Let me also show you this CSS example:

```
<p>Formatting with <span style="text-decoration:overline;">
  CSS</span> and the span element.</p>
```

Here, the text is preceded by a stroke that has been implemented via style statements directly in the HTML tag `span` with the HTML attribute `style`.



Figure 4.49 The `` Element Has No Default Formatting; Besides Designing with CSS, It Can Also Be Used to Identify Unique Elements.

However, as with the `div` element, you should only use this element if there's no semantically more suitable HTML element available.

Example of All Text Markup Elements

Once again, note that you can find an example with all the text markup elements demonstrated here on the web page for the book (www.rheinwerk-computing.com/5695/) and at https://html-examples.pronix.de/examples/chapter004/4_4/index.html.

4.5 Related Topic: Character Encoding

What follows here isn't a treatise on character encoding in general, but rather in the context of HTML documents—especially on how you can avoid special characters, such as German umlauts, from getting displayed as cryptic characters. If you consider the following two points, there shouldn't be any problem:

- In the HTML document, you need to specify the character encoding of the document in the head data between `<head>` and `</head>`, as we've been doing in the book so far with `<meta charset="UTF-8">`. Unless you have a specific reason, you should always use the UTF-8 value for the `charset` attribute.

- However, it doesn't suffice to specify the character encoding in the HTML document as it must also be saved in this encoding using the editor of your choice. Consequently, if you've specified UTF-8 as the character encoding in the HTML document, you must also save the document with the UTF-8 encoding. With most editors, you no longer have to bother about this. Nevertheless, it should be briefly mentioned here.

4.5.1 From Bytes to Character Encoding

The smallest unit, the bit, will be skipped here because you don't need to go so deep into detail at this point. The byte unit is quite sufficient for this purpose. When the computer reads a file or data into the main memory, it's basically just bytes that have a certain value. The value of a single byte results from the states of the individual bits. Let's use a byte with the value 68 (incidentally, with the bit value 1.000.100) as an example.

To create a human-readable character from this byte with the value 68, a convention is needed that describes which byte value corresponds to which representable character. For this purpose, a translation table (also referred to as encoding table) is used for encoding bytes.

4.5.2 From ASCII to ISO-8859

You know that an encoding table is responsible for turning a byte into a readable character. The first type of such a character set was introduced with the ASCII encoding and the EBCDIC encoding, with which 128 different states can be represented on 7 bits. ASCII encoding has become established in common practice. In the ASCII encoding table, the first 32 characters are pure control characters, and the actual characters are stored in the character set between 32 and 127. A look at the ASCII encoding table shows that the value 68 corresponds to the capital letter D.

The 8th bit was initially used only for error-correction purposes (parity bit) for communication lines or other control tasks. Because there was no space left in the ASCII character set between the values 32 and 127 for language-specific characters (e.g., umlauts), the 8th bit was used to extend the character set. At this point, the Babylonian character confusion also arose because different developers wrote their own "8th-bit-codes." IBM PCs and English MS-DOS systems used codepage 435, for example. In Germany, codepage 850 was used for Western European characters.

Newer standards such as ISO-8859 also use 8 bits. Here, several character set tables were developed at once. For example, ISO-8859-1 (or Latin-1) represents the Western European languages. The first 127 characters were taken over from the ASCII encoding. In the

values between 128 and 255, many special characters and important characters from different European languages were implemented (with the German umlauts, the Spanish tilde character, or the French accent characters).

So, theoretically, you can use the ISO-8859-1 character set for the HTML document:

```
<meta charset="iso-8859-1">
```

While in theory, you can use any character set for `charset`, you should keep in mind that not every web browser understands all character sets. If you use a more widely used character set, you have a better chance that a web browser in distant countries will be able to do something with it.

Microsoft had also added its own variation to the ISO-8859-1 encoding with codepage 1252. After all, code page 1252 already contained the euro sign. ISO-8859-1, on the other hand, doesn't recognize the euro sign because at the time this table was created, the euro didn't even exist. The euro sign was added by the ISO only later with ISO-8859-15. Now the situation here is that ISO-8859-1 doesn't recognize the euro sign, while ISO-8859-15 and codepage 1252 do know it, but the value in the encoding table is again different. Fortunately, today you don't need to deal with the different character sets of a language. The description of the ISO 8859 standards here serves only as background information on the subject.

The current HTML specification uses the Unicode UTF-8 character set with `charset="UTF-8"`.

4.5.3 Beyond the Byte Boundary with Unicode

The preceding provided a good impression of the confusion regarding the different character encodings. Note, however, that this was only about the Western European character set, and I haven't really gone into detail yet. The fact is that character encoding can be relatively complex if you pack everything into a byte and then want to use different characters from different cultures. To bring all characters under one hood, the Unicode system was introduced.

The Unicode character set can be used to represent all human-made characters. In purely theoretical terms, more than four billion characters could be used with 32 bits per character—in practice, Unicode is limited to about one million code points. UTF-8 is the 8-bit encoding of Unicode, which is also backward compatible with ASCII encoding. A character can contain between one and four 8-bit words. UTF-8 is now a uniform standard. For example, many operating systems use UTF-8 by default, and UTF-8 is also being used increasingly in web development with HTML to represent language-specific characters, where it more and more replaces the use of HTML entities (Section 4.6).

More Information Online

I could write much more on this topic, especially Unicode, but this would go beyond the scope of this book. For more information, visit <http://r12a.github.io/scripts/tutorial/> and <https://home.unicode.org/>. You can also find the characters of the Unicode encoding at www.unicode.org/charts/.

4.6 Character Entities in HTML

While the importance of character entities in HTML has diminished considerably with the gradual spread of Unicode (especially UTF-8), I should still touch on them briefly here because there are always reasons to use them. In Section 4.5, you learned about different character sets, and, by now, you know how to specify the character set used as a `<meta>` tag in the HTML document head with `charset`. For example, if you’ve specified ISO-8859-1 or ISO-8859-15 as the character set and want to use the word `shalom` (שלום) in Hebrew characters, you’re likely to be unsuccessful:

```
<meta charset="iso-8859-1">
...
<p>Shalom: שלום</p>
```

The output is likely to be a cluster of cryptic characters instead of שלום. The simplest solution would be to change the character set to UTF-8 via

```
<meta charset="UTF-8">
```

There might also be a different problem with this example: How do you type the word שלום in your editor? If you don’t happen to have a Hebrew keyboard in front of you or a virtual keyboard with Hebrew characters, the simple and quick solution might be to use the character entities of HTML. This is how you write the word “Shalom” in Hebrew using character entities:

```
<p>Shalom: <bdo>&#1501;&#1503;&#1500;&#1513;</bdo></p>
```

4.6.1 Structure of a Character Entity in HTML

As you’ve seen before from the four Hebrew characters, an HTML entity starts with the `&` character and ends with the semicolon. Now you have two options to arrange the sign:

- **Numeric entities**
You specify the form with `&#nnn;`. Here, `nnn` stands for the encoding of the character. This form is used when it isn’t possible to enter the character via the keyboard. The

notation can also be in the form of `&#xhhh;`, where `xhhh` is the hexadecimal value for the character. The notation without `x` is the decimal notation.

- **Named entities**
This is an easier-to-remember name that has been agreed on for the character. You may have already seen examples with `<`; (`lt` = *less than*) or `>`; (`gt` = *greater than*) where people prefer to use these named entities. Alternatively, you can use the numeric entity instead of the named entities. For example, with `<`;, `<`;, and `<`;, you would use the `<` sign (less-than sign) three times.

Masking HTML-Specific Characters

Especially if you use special characters in your body text that are part of the HTML syntax, you should mask these characters by using the appropriate entity. For example, the following line is likely to cause display problems in a web browser:

```
<p>Mexico City<Tokyo and Mumbai>London</p>
```

The web browsers would only output Mexico City-London here, because the area between `<` and `>` is considered an HTML element (even if it’s wrong). Although you could solve this problem with a blank line in between, you should use the appropriate entity for this, to be on the safe side. This is where the named entity comes in handy:

```
<p>Mexico City&lt;Tokyo and Mumbai&gt;London</p>
```

The ampersand character `&` belongs to it as well and should be used via the string `&`; in the continuous text.

In addition, if you want to use the double quote within HTML attributes, you should replace `"` with `"`;, such as the following:

```

```

In the `alt` attribute, `"` was used as a masking character for `"`. If you used `"` instead of the named entity `"`; here, the area in between would probably be “swallowed” by the web browser.

More Unicode Numbers

You can find even more Unicode numbers for a desired character at www.unicode.org/charts.

4.7 Summary

In this chapter, you learned a lot about the semantic use of various HTML elements. Roughly summarized, you’ve learned the following in this chapter:

- How to use the `<section>`, `<article>`, `<aside>`, and `<nav>` elements to divide an HTML document into meaningful parts (or sections)
- How to use headings with the elements `<h1>` to `<h6>` and the new section elements `<section>`, `<article>`, `<aside>`, and `<nav>` to affect the content structure of headings
- How to use a header with `<header>` and a footer with `<footer>` in an HTML document
- How to use the `<main>` element to set the main content of a web page
- Which HTML elements are available to divide or group plain text content into paragraphs
- What semantic HTML is and how you can structure a semantic web page
- How to logically mark up text, individual letters, words, or parts of sentences to give them semantic meaning
- That the HTML elements for text markup aren't used for formatting web pages, but that this is done via CSS

Contents

Preface	25
1 Introduction to the HTML Universe	31
1.1 Is This Book Even Intended for Me?	31
1.2 Different Types of Websites	32
1.2.1 Web Presence	32
1.2.2 Blog/Online Magazine/Portfolio	33
1.2.3 E-Commerce Websites: Stores without Opening Hours	35
1.2.4 Landing Page/Microsite	36
1.2.5 Web Platform: Building Your Own Social Network	36
1.2.6 Web Apps	37
1.3 Dynamic and Static Websites	37
1.3.1 Static Websites	38
1.3.2 Dynamic Websites	39
1.4 Languages for Designing and Developing on the Web	41
1.4.1 HTML: Text-Based Hypertext Markup Language	41
1.4.2 CSS: Design Language	42
1.4.3 JavaScript: Client-Side Scripting Language of the Web Browser	43
1.4.4 Server-Side Scripting Languages and Databases	43
1.5 What Do I Need to Get Started?	44
1.5.1 HTML Editor for Writing HTML Documents	44
1.5.2 Web Browser for Displaying the Website	46
1.5.3 Step by Step: Creating a Web Page and Viewing It in the Web Browser	47
1.5.4 Checking Written HTML	49
1.5.5 Good Reasons for Validating the HTML Code Anyway	52
1.6 Conventions Used in This Book	53
1.7 Summary	53

2	Basic Structure of HTML and HTML Documents	55
2.1	Syntax and Structure of HTML and HTML Documents	55
2.1.1	How to Structure a Document in HTML	55
2.1.2	Viewing the Tree Structure Using the Document Object Model Inspector	58
2.1.3	HTML Tags and HTML Elements	59
2.1.4	Nesting HTML Elements and the Hierarchical Structure	60
2.1.5	Avoiding Incorrect Nesting of HTML Elements	61
2.1.6	Omitting the End Tag of an HTML Element	62
2.1.7	Standalone HTML Tags without End Tags	63
2.1.8	Additional HTML Attributes for HTML Elements	64
2.1.9	Using Comments in HTML Documents	65
2.2	A Simple HTML Document Framework	65
2.2.1	HTML Document Type: <!doctype>	66
2.2.2	Beginning and Ending an HTML Document: <html>	67
2.2.3	Head of an HTML Document: <head>	67
2.2.4	Visible Part of an HTML Document: <body>	68
2.3	Summary	68
3	Head Data of an HTML Document	69
3.1	Overview of HTML Elements for the Head	69
3.2	<title>: Heading of the HTML Page	70
3.3	Related Topic: Naming Convention and Referencing	72
3.3.1	Valid and Good File Names for an HTML Document	72
3.3.2	Valid Directory Names and Meaningful Directory Structures	73
3.3.3	Writing a Reference to a Data Source	73
3.4	Defining the Base URL of a Web Page Using <base>	76
3.5	Referencing an External Document via <link>	78
3.6	Writing Document-Wide CSS Styles Using <style>	81
3.7	Including Scripts in Web Pages Using <script>	82
3.8	Metadata for the Document Using <meta>	85
3.8.1	The Most Commonly Used Metadata	85
3.8.2	Setting the Viewport	87
3.8.3	Specifying Useful Metadata for a Web Crawler	88
3.8.4	Useful Metadata for Search Engines	88

3.8.5	Useful Metadata for the Web Browser	90
3.8.6	Using General Metadata	91
3.8.7	My Recommendation: This Metadata Belongs in the Basic HTML Framework	92
3.8.8	HTML Attributes for the <meta> Element	92
3.9	Summary	93
4	The Visible Part of an HTML Document	95
4.1	HTML Elements for Structuring Pages	95
4.1.1	Using <body>: The Displayable Content Section of an HTML Document	96
4.1.2	Introducing the Section Elements of HTML	96
4.1.3	Using Headings with the HTML Elements from <h1> to <h6>	104
4.1.4	Creating a Header Using <header> and a Footer Using <footer>	108
4.1.5	Marking Contact Information Using <address>	110
4.2	HTML Elements for Structuring Text	111
4.2.1	Adding Text Paragraphs Using <p>	112
4.2.2	Forcing Line Breaks Using 	113
4.2.3	Adding Optional Line Breaks Using <wbr>	114
4.2.4	Forcing Spaces and Preventing Wrapping Using " "	115
4.2.5	Adding a Topic-Based Separation Using <hr>	116
4.2.6	Adding Paragraphs or Citations Using <blockquote>	117
4.2.7	Defining a General Section Using <div>	118
4.2.8	Using <main>: An HTML Element for the Main Content	120
4.2.9	Labeling Content Separately Using <figure> and <figcaption>	121
4.2.10	Creating Unordered Lists Using and 	122
4.2.11	Creating Ordered Lists Using and 	123
4.2.12	Reversing the Numbering of an Ordered List	124
4.2.13	Changing the Numbering of an Ordered List	124
4.2.14	Nesting Lists within Each Other	125
4.2.15	Creating a Description List Using <dl>, <dt>, and <dd>	128
4.3	Using Semantic HTML	130
4.3.1	HTML without a Precise Structure	130
4.3.2	Generic Structuring Using <div>	132
4.3.3	Semantic Structuring Using the Elements Provided in HTML	135
4.3.4	What's the Use of Those Semantic HTML Elements?	138
4.4	HTML Elements for Text Markups	138
4.4.1	Marking Up Abbreviations or Acronyms Using <abbr>	140

4.4.2	Marking Up Text as the Source of a Working Title Using <cite>	140
4.4.3	Marking Up Computer Code Representation Using <code> and <pre>	141
4.4.4	Keyboard Input Using <kbd> and Program Output Using <samp>	142
4.4.5	Marking Up Text as a Definition Using <dfn>	143
4.4.6	Marking Up Text as a Variable Using <var>	144
4.4.7	Changing the Text Direction Using <bdo> and <bdi>	144
4.4.8	Emphasizing Text Using , , <i>, and 	145
4.4.9	Highlighting Text Using <mark>	146
4.4.10	Placing Text between Quotes Using <q>	147
4.4.11	Underlining or Crossing Out Text Using <u> and <s>	148
4.4.12	Marking Changes of Text Using <ins> and 	150
4.4.13	Displaying Text as Superscript or Subscript Using <sup> and <sub> ...	151
4.4.14	Marking Dates and Times Using <time>	151
4.4.15	Marking the Small Print Using <small>	155
4.4.16	Using <ruby>, <rp>, and <rt> for Annotations about Pronunciation ...	155
4.4.17	Grouping Ranges of Individual Text Passages Using 	156
4.5	Related Topic: Character Encoding	157
4.5.1	From Bytes to Character Encoding	158
4.5.2	From ASCII to ISO-8859	158
4.5.3	Beyond the Byte Boundary with Unicode	159
4.6	Character Entities in HTML	160
4.6.1	Structure of a Character Entity in HTML	160
4.7	Summary	161
5	Tables and Hyperlinks	163
5.1	Structuring Data in a Table	163
5.1.1	A Simple Table Structure Using <table>, <tr>, <td>, and <th>	164
5.1.2	Combining Columns or Rows Using “colspan” or “rowspan”	166
5.1.3	HTML Attributes for the Table Elements	169
5.1.4	Structuring Tables Using <thead>, <tbody>, and <tfoot>	169
5.1.5	Grouping Columns of a Table Using <colgroup> and <col>	172
5.1.6	Labeling Tables Using <caption> or <figcaption>	174
5.2	Electronic References (Hyperlinks) Using <a>	177
5.2.1	Inserting Links to Other HTML Documents on Your Own Website	178
5.2.2	Inserting Links to Other Websites	181
5.2.3	Opening Links with the “target” Attribute in a New Window	182
5.2.4	Email Links with “href=mailto: . . .”	183

5.2.5	Setting Links to Other Types of Content	185
5.2.6	Adding Download Links Using the “download” Attribute	186
5.2.7	Setting Links to Specific Parts of a Web Page	188
5.2.8	Creating Links to Phone Numbers	191
5.2.9	HTML Attributes for the HTML Element <a>	192
5.3	Summary	193
6	Graphics and Multimedia	195
6.1	Embedding Images Using 	196
6.1.1	Adding Images to an HTML Document	196
6.1.2	Specifying the Height and Width of a Graphic	200
6.1.3	Labeling Images Using <figure> and <figcaption>	202
6.1.4	HTML Attributes for the HTML Element 	204
6.2	Creating Link-Sensitive Graphics (Image Maps)	204
6.2.1	Use HTML Attributes for the HTML Element <area>	207
6.2.2	Referencing Defined Areas of the HTML Element <area>	208
6.2.3	HTML Attributes of <area>	208
6.3	Loading the Appropriate Image Using <picture>	210
6.3.1	HTML Attributes of <source>	211
6.3.2	Multiple Image Sources with the HTML Attribute “srcset”	212
6.4	Adding an Icon for the Website (Favicon)	213
6.5	Using Vector Graphics in HTML Documents	214
6.5.1	Adding SVG as a Graphic Reference Using 	215
6.5.2	Embedding SVG Directly into the Web Page Using <svg>	216
6.5.3	SVG Tags for Vector Graphics	217
6.5.4	Overview of Graphical SVG Elements	217
6.5.5	Further Notes on Using SVG	219
6.5.6	Mathematical Formulas Using MathML	219
6.6	Drawing Graphics Using <canvas>	221
6.7	Playing Videos Using the HTML Element <video>	222
6.7.1	HTML Attributes for the HTML Element <video>	224
6.7.2	Adding Subtitles to a Video Using <track>	225
6.7.3	Playing Videos via YouTube	228
6.8	Playing Audio Files Using the HTML Element <audio>	229
6.8.1	HTML Attributes for the HTML Element <audio>	230
6.9	Including Other Active Content	231
6.9.1	HTML Element <embed>	232

6.9.2	HTML Element <object>	232
6.9.3	HTML Element <iframe>	233
6.10	Summary	235
7 HTML Forms and Interactive Elements		237
7.1	Defining a Space for Forms	238
7.2	HTML Input Fields for Forms	239
7.2.1	A Single-Line Text Input Field Using <input type="text">	240
7.2.2	A Password Input Field Using <input type="password">	240
7.2.3	A Multiline Text Input Field Using <textarea>	241
7.2.4	A Selection List or Dropdown List Using <select>	242
7.2.5	Creating a Group of Radio Buttons Using <input type="radio">	244
7.2.6	Adding a Text Label Using <label>	245
7.2.7	Using Checkboxes via <input type="checkbox">	245
7.2.8	Using Fields for File Uploads via <input type="file">	246
7.2.9	Overview of Various Buttons	247
7.2.10	Using a Hidden Input Field via <input type="hidden">	248
7.2.11	Writing Form Fields outside of <form>...</form>	248
7.2.12	Multiple Submit Buttons for Different URLs	249
7.3	Special Types of Input Fields	250
7.3.1	An Input Field for Colors Using <input type="color">	251
7.3.2	An Input Field for a Date Using <input type="date">	252
7.3.3	An Input Field for a Time Using <input type="time">	252
7.3.4	Input Fields for Date and Time	253
7.3.5	Input Fields for the Month and the Week	254
7.3.6	An Input Field for Searches Using <input type="search">	254
7.3.7	An Input Field for Email Addresses Using <input type="email">	255
7.3.8	An Input Field for a URL Using <input type="url">	255
7.3.9	An Input Field for Phone Numbers Using <input type="tel">	256
7.3.10	An Input Field for Numbers Using <input type="number">	256
7.3.11	An Input Field for Numbers of a Certain Range	256
7.3.12	Outputting Values and Calculations Using <output>	256
7.4	The HTML Attributes for Input Fields	257
7.4.1	Setting the Input Focus Using the HTML Attribute "autofocus"	258
7.4.2	(De)activating Autocompletion Using the "autocomplete" Attribute	258

7.4.3	A List of Suggestions for Using the HTML Attribute "list" and <datalist>	259
7.4.4	Specifying Minimum and Maximum Values and the Step Size	259
7.4.5	Selecting or Entering Multiple Values Using "multiple"	260
7.4.6	Regular Expressions for Input Fields Using "pattern"	260
7.4.7	A Placeholder for an Input Field Using "placeholder"	260
7.4.8	Defining an Input as Required Using the "required" Attribute	261
7.4.9	Controlling Error Messages for Input Fields	261
7.5	Other Useful Helpers for Input Fields	263
7.5.1	Disabling Form Elements Using the HTML Attribute "disabled"	264
7.5.2	Permitting Read-Only for Input Fields Using the "readonly" Attribute	265
7.5.3	Useful Keyboard Shortcuts and Tab Sequence for Input Fields	265
7.5.4	Grouping Form Elements Using <fieldset> and <legend>	266
7.5.5	Progress Display via <progress>	267
7.5.6	Visualizing Values Using <meter>	268
7.6	Sending Form Data Using PHP	268
7.6.1	Transferring the Data from the Web Browser for Further Processing	269
7.6.2	The "POST" Method	271
7.6.3	The "GET" Method	271
7.6.4	Processing the Data Using a PHP Script	272
7.7	Interactive HTML Elements	275
7.7.1	Expanding/Collapsing Content Using <details> and <summary>	275
7.7.2	A Dialog Box via <dialog>	276
7.8	Summary	277
8 Introduction to Cascading Style Sheets		279
8.1	The Story of CSS	280
8.2	The Basic Principle of Using CSS	281
8.2.1	Structure of a CSS Rule	283
8.2.2	Declaring a Selector	283
8.2.3	Using Comments for CSS Code	284
8.2.4	A Few Notes on Formatting CSS Code	285
8.3	Integrating CSS into HTML	285
8.3.1	Style Statements Directly in the HTML Tag Using the HTML Attribute "style"	286

8.3.2	Style Statements in the Document Head Using the HTML Element <style>	287
8.3.3	Integrating Style Statements from an External CSS File Using <link>	288
8.3.4	Combining CSS Rules in the Head Section and in External CSS Files	289
8.3.5	Recommendation: You Should Separate HTML and CSS	291
8.3.6	Testing Alternate Stylesheets during Development	291
8.3.7	Integrating Style Statements from an External CSS File Using “@import”	293
8.3.8	Media-Specific Stylesheets for Specific Output Devices	293
8.3.9	Media-Specific Stylesheets with CSS	295
8.4	Analyzing CSS in the Web Browser	295
8.5	Summary	296
9	The Selectors of CSS	297
9.1	The Simple Selectors of CSS	298
9.1.1	Addressing HTML Elements Using the Type Selector	299
9.1.2	Addressing HTML Elements Using a Specific Class or ID	301
9.1.3	Universal Selector: Addressing All Elements in a Document	308
9.1.4	Addressing Elements Based on Attributes Using the Attribute Selector	310
9.1.5	An Attribute Selector for Attributes with a Specific Value	312
9.1.6	Attribute Selector for Attributes with a Specific Partial Value	315
9.1.7	CSS Pseudo-Classes: The Selectors for Specific Features	318
9.1.8	The Convenient Structural Pseudo-Classes in CSS	322
9.1.9	Other Useful Pseudo-Classes	329
9.1.10	Pseudo-Elements: The Selectors for Nonexistent Elements	330
9.2	Combinators: Concatenating the Selectors	332
9.2.1	The Descendant Combinator (E1 E2)	334
9.2.2	The Child Combinator (E1 > E2)	335
9.2.3	The Adjacent Sibling Combinator (E1 + E2)	337
9.2.4	The General Sibling Combinator (E1 ~ E2)	338
9.3	Recommendation: How to Use Efficient and Simple CSS	340
9.3.1	How to Write Well Performing CSS	340
9.3.2	Recommendation: Keep the CSS Code as Simple as Possible	342
9.4	Summary	343

10	Inheritance and Cascading	345
10.1	The Principle of Inheritance in CSS	345
10.1.1	Be Cautious When Using Relative Properties	349
10.1.2	Not Everything Gets Inherited	350
10.1.3	Enforcing Inheritance Using “inherit”	350
10.1.4	Restoring the Default Value of a CSS Feature (“initial”)	352
10.1.5	Forcing Inheritance or Restoring a Value (“unset”)	352
10.1.6	Forcing Inheritance or Restoring Values for All Properties	353
10.2	Understanding the Control System for Cascading	354
10.2.1	The Origin of a Stylesheet	354
10.2.2	Increasing the Priority of a CSS Feature Using “!important”	355
10.2.3	Sorting by Importance and Origin	356
10.2.4	Sorting by Weighting the Selectors (Specificity)	357
10.2.5	Summary of the Cascading Rules System	361
10.2.6	Analyzing the Cascading in the Browser	362
10.3	Related Topic: Passing Values to CSS Features	363
10.3.1	Different Units of Measurement in CSS	363
10.3.2	Character Strings and Keywords as Values for CSS Features	365
10.3.3	Many Ways of Using a Color in CSS	366
10.3.4	Angular Dimensions in CSS	372
10.3.5	Passing Values via Short Notation to a CSS Feature	373
10.4	Summary	374
11	The Box Model of CSS	375
11.1	Classic Box Model of CSS	376
11.1.1	Specifying the Content Area Using “width” and “height”	376
11.1.2	Specifying the Inner Spacing Using “padding”	378
11.1.3	Creating the Border Using “border”	379
11.1.4	Setting Up the Outer Margin Using “margin”	379
11.1.5	Collapsing Margins	381
11.1.6	Determining the Total Width and Total Height of a Box	385
11.2	Newer Alternate Box Model of CSS	386
11.2.1	Using the “box-sizing” Box Model	388
11.2.2	Using the Alternate Box Model	388
11.3	Analyzing the Box Model in the Browser	392
11.4	Box Model for Inline Elements	393

11.5 Designing Boxes	393
11.5.1 Adding and Designing a Border Using the “border” Property	393
11.5.2 Setting a Background Color Using “background-color”	397
11.5.3 Using Background Images	397
11.5.4 Making Boxes Transparent	405
11.5.5 Adding a Gradient	406
11.5.6 Adding a Shadow Using the “box-shadow” Feature	409
11.5.7 Adding Round Corners Using the CSS Feature “border-radius”	411
11.6 Related Topic: Web Browser Prefixes (CSS Vendor Prefixes)	413
11.7 Summary	416

12 CSS Positioning 417

12.1 Positioning via CSS Feature “position”	417
12.1.1 Normal Positioning (“position: static”)	418
12.1.2 Positioning Elements Using “top”, “right”, “bottom”, and “left”	420
12.1.3 Relative Positioning (“position: relative”)	421
12.1.4 Absolute Positioning (“position: absolute”)	422
12.1.5 Fixed Positioning (“position: fixed”)	426
12.1.6 Sticky Positioning (“position: sticky”)	429
12.2 Controlling Stacking Using “z-index”	431
12.3 Floating Boxes for Positioning via “float”	434
12.3.1 Terminating the Float	438
12.3.2 Combining Floats into One Entity	440
12.4 Flexible Boxes of CSS	443
12.4.1 Aligning the Flexbox	443
12.4.2 Setting the Flexibility of the Flexbox	451
12.4.3 Determining the Order of the Boxes	454
12.5 Summary	454

13 Creating Responsive Layouts with CSS 457

13.1 Basic Theoretical Knowledge of Responsive Web Design	457
13.1.1 Using Specific Media Types	458
13.1.2 Media Queries for Media Features	461
13.1.3 Integrating and Applying Media Queries for Media Features	461
13.1.4 Basic Structure of a Media Feature Query	462

13.1.5 Which Media Features Can Be Queried?	464
13.1.6 Crucially Important: The Viewport for Mobile Devices	465
13.1.7 Use “em” Instead of Pixels for a Layout Break in Media Queries	469
13.1.8 Layout Breaks (Breakpoints)	471
13.1.9 No More Math Games Thanks to “box-sizing: border-box;”	472
13.1.10 What Happens to Web Browsers That Don’t Understand Media Queries?	472
13.2 Let’s Create a Simple Responsive Layout	472
13.2.1 Let’s Create the Basic Framework Using HTML	473
13.2.2 Setting General CSS Features	474
13.2.3 What Should I Use as a Basic Version without Media Queries: Mobile First?	475
13.2.4 Setting the Layout Break (Breakpoint)	480
13.2.5 Adding More Layout Breaks	482
13.2.6 Customizing the Main Content	487
13.3 Even More Flexible Elements	489
13.3.1 Use Relative Font Sizes instead of Pixels	489
13.3.2 Making Images Responsive	489
13.3.3 Flexible Images in Maximum Possible Width	493
13.3.4 Hiding Images Entirely	495
13.3.5 Loading the Right Image for the Screen Width: <picture>	496
13.3.6 Using Area-Covering Images	498
13.4 CSS Grid Layout	501
13.4.1 Creating a Grid for the Content	501
13.4.2 Placing Elements in the Grid	504
13.4.3 Layout Changes Made Easy	510
13.4.4 Spacing between Grid Lines	511
13.4.5 Checking the Grid in the Web Browser	512
13.5 Changing the Behavior of HTML Elements Using “display”	513
13.5.1 “display: block”, “display: inline”, and “display: inline-block”	513
13.5.2 Hiding Elements Using “display:none”	515
13.5.3 Further Values for “display”	516
13.6 Calculations Using CSS and the “calc()” Function	516
13.7 Summary	519

14 Styling with CSS 521

14.1 Designing Texts with CSS	521
14.1.1 Selecting Fonts via “font-family”	522

14.1.2	Providing Fonts via Web Fonts: “@font-face”	526
14.1.3	Using Icons via Icon Fonts	532
14.1.4	Setting the Font Size Using “font-size”	536
14.1.5	Italic and Bold Fonts via “font-style” and “font-weight”	543
14.1.6	Creating Small Caps Using “font-variant”	544
14.1.7	Defining Line Spacing via “line-height”	545
14.1.8	A Short Notation for Font Formatting Using “font”	546
14.1.9	Specifying Letter and Word Spacing via “letter-spacing” and “word-spacing”	547
14.1.10	Setting the Text Alignment Using “text-align”	548
14.1.11	Setting the Vertical Alignment via “vertical-align”	550
14.1.12	Indenting Text Using “text-indent”	551
14.1.13	Underlining Text and Striking Text Through Using “text-decoration”	552
14.1.14	Uppercase and Lowercase Text via “text-transform”	553
14.1.15	Adding Shadow to Text via “text-shadow”	554
14.1.16	Splitting Text into Multiple Columns Using “column-count”	555
14.2	Designing Lists with CSS	557
14.2.1	Customizing Bullet Points Using “list-style-type”	557
14.2.2	Using Images as Bullets via “list-style-image”	559
14.2.3	Positioning Bulleted Lists via “list-style-position”	560
14.2.4	Short Notation “list-style” for Designing Lists	560
14.2.5	Creating Navigation and Menus via Lists	561
14.3	Designing Appealing Tables with CSS	566
14.3.1	Creating Fixed-Width Tables	566
14.3.2	General Recommendation: Designing Appealing Tables with CSS	567
14.3.3	Collapsing Borders for Table Cells Using “border-collapse”	568
14.3.4	Setting the Spacing between Cells via “border-spacing”	569
14.3.5	Displaying Empty Table Cells Using “empty-cells”	570
14.3.6	Positioning Table Captions via “caption-side”	571
14.4	Adjusting Images and Graphics Using “width” and “height”	571
14.5	Transforming Elements with CSS	574
14.5.1	Scaling HTML Elements via “transform: scale()”	575
14.5.2	Rotating HTML Elements Using “transform: rotate()”	576
14.5.3	Skewing HTML Elements Using “transform: skew()”	577
14.5.4	Moving HTML Elements Using “transform: translate()”	577
14.5.5	Combining Different Transformations	578
14.5.6	Other HTML Elements	579
14.6	Creating Transitions with CSS	580

14.7	Styling HTML Forms with CSS	581
14.7.1	Neatly Structuring an HTML Form	582
14.7.2	Aligning Form Elements with CSS	584
14.7.3	Designing Form Elements with CSS	587
14.8	Summary	590
15	Testing and Organizing	591
15.1	Web Browser Tests: What’s Possible?	591
15.1.1	Validating HTML and CSS	592
15.1.2	Which Browsers Are Visitors Currently Using?	592
15.1.3	CSS Web Browser Test	593
15.1.4	HTML5 Web Browser Test	594
15.1.5	Can I Use That?	595
15.1.6	Feature Query Using the “@supports” Rule	596
15.2	Viewing Websites in Different Sizes	596
15.3	Setting Up a Central Stylesheet	598
15.3.1	Combining Everything Back into One File to Shorten the Load Time	600
15.4	CSS Reset or Normalization?	600
15.4.1	Built-In Style Presets of the Web Browser and CSS Reset	600
15.4.2	Normalization: The Alternative to CSS Reset	602
15.5	Summary	603
16	The CSS Preprocessor Sass and SCSS	605
16.1	Sass or SCSS Syntax	605
16.2	From Sass/SCSS to CSS	606
16.3	Installing and Setting Up Sass	607
16.3.1	Online CSS Preprocessor without Installation	607
16.3.2	Setting Up Sass Using Visual Studio Code	608
16.3.3	Installing Sass for the Command Line	610
16.4	Using Variables with Sass	611
16.5	Nesting with Sass	613
16.6	Mixins (“@mixin”, “@include”)	615
16.7	Extend (“@extend”)	618

16.8 Media Queries and “@content”	621
16.9 Operators	624
16.10 Adjusting Colors and Brightness	625
16.11 Sass Control Structures	628
16.12 Functions “@function”	632
16.13 “@import”	633
16.14 Comments	634
16.15 Summary	635
17 A Brief Introduction to JavaScript	637
17.1 JavaScript in Web Development	638
17.2 Writing and Executing JavaScript Programs	640
17.2.1 Integrating a JavaScript File in an HTML File	641
17.2.2 Writing JavaScript within HTML	643
17.2.3 Position of JavaScript and Its Execution in the HTML Document	644
17.2.4 Attributes for Manipulating the Load Behavior of JavaScript (“async”, “defer”)	645
17.2.5 The <noscript> Element for No JavaScript	645
17.3 JavaScript Output	646
17.3.1 Standard Dialogs (and Input Dialog)	646
17.3.2 Outputting to the Console	647
17.3.3 Outputting to the Website	649
17.3.4 Running JavaScript without a Web Browser	651
17.3.5 Annotating JavaScript Code with Comments	652
17.4 Using Variables in JavaScript	652
17.4.1 Defining Constants	655
17.4.2 Strict Mode Using “use strict”	656
17.5 Overview of JavaScript Data Types	657
17.5.1 Number Data Type (Numbers)	657
17.5.2 String Data Types (Strings)	658
17.5.3 Template Strings	660
17.5.4 Boolean Data Type	660
17.5.5 Undefined and Null Data Types	661
17.5.6 Objects	662
17.5.7 Converting Data Types	662
17.6 Arithmetic Operators for Calculation Tasks in JavaScript	663

17.7 Conditional Statements in JavaScript	665
17.7.1 “true” or “false”: Boolean Truth Value	666
17.7.2 Using the Various Comparison Operators in JavaScript	667
17.7.3 Using the “if” Branch	668
17.7.4 Using the Selection Operator	669
17.7.5 Logical Operators	669
17.7.6 Multiple Branching via “switch”	670
17.8 Multiple Repetitions of Program Statements via Loops	672
17.8.1 Increment and Decrement Operators	672
17.8.2 The Header-Controlled “for” Loop	673
17.8.3 The Header-Controlled “while” Loop	674
17.8.4 The Footer-Controlled “do-while” Loop	674
17.8.5 Ending the Statement Block Using “break”	675
17.8.6 Jumping to the Start of the Loop via “continue”	675
17.9 Summary	676
18 Arrays, Functions, and Objects in JavaScript	677
18.1 Functions in JavaScript	677
18.1.1 Different Ways to Define a Function in JavaScript	678
18.1.2 Calling Functions and Function Parameters	680
18.1.3 Return Value of a Function	683
18.1.4 The Scope of Variables in a Function	683
18.1.5 Defining Functions in Short Notation (Arrow Functions)	686
18.1.6 Using a Function in a Web Page	687
18.2 Arrays	689
18.2.1 Accessing the Individual Elements in the Array	690
18.2.2 Multidimensional Arrays	691
18.2.3 Adding or Removing New Elements in an Array	692
18.2.4 Sorting Arrays	697
18.2.5 Searching within Arrays	698
18.2.6 Additional Methods for Arrays	699
18.3 Strings and Regular Expressions	700
18.3.1 Useful Functions for Strings	700
18.3.2 Applying Regular Expressions to Strings	701
18.4 Object-Oriented Programming in JavaScript	702
18.4.1 What Exactly Are Objects?	702
18.4.2 Creating Objects via Constructor Functions	703
18.4.3 Creating Objects via the Class Syntax	704

18.4.4	Accessing the Object Properties and Methods: Setters and Getters	705
18.4.5	The Keyword “this”	708
18.5	Other Global Objects	709
18.5.1	The Top Object “Object”	709
18.5.2	Objects for the Primitive Data Types: Number, String, and Boolean	709
18.5.3	“Function” Object	711
18.5.4	“Date” Object	711
18.5.5	“Math” Object	711
18.5.6	“Map” Object	712
18.5.7	“Set” Object	712
18.6	Summary	713
 19 Changing Web Pages Dynamically		715
19.1	Introduction to the DOM of an HTML Document	715
19.2	The “document” Object	717
19.3	DOM Programming Interface	717
19.4	Accessing Elements in the DOM	718
19.4.1	Finding an HTML Element with a Specific “id” Attribute	719
19.4.2	Finding HTML Elements with a Specific Tag Name	720
19.4.3	Finding HTML Elements with a Specific “class” Attribute	723
19.4.4	Finding HTML Elements with a Specific “name” Attribute	724
19.4.5	Using “querySelector()” and “querySelectorAll()”	725
19.4.6	Other Object and Property Collections	727
19.5	Changing an HTML Element, an Attribute, or the Style	730
19.5.1	Changing the Content of HTML Elements Using “innerHTML”	730
19.5.2	Changing the Value of an HTML Attribute	732
19.5.3	Changing the Style (CSS) of an HTML Element	733
19.6	Responding to JavaScript Events	735
19.7	Handling the Events Using the Event Handler	736
19.7.1	Setting Up an Event Handler as an HTML Attribute in the HTML Element	737
19.7.2	Setting Up Event Handlers as a Property of an Object	737
19.7.3	Setting Up an Event Handler via “addEventListener()”	738
19.8	Overview of Common JavaScript Events	740
19.8.1	The JavaScript Events of the UI (Window Events)	740
19.8.2	JavaScript Events That Can Occur in Connection with the Mouse	742

19.8.3	JavaScript Events for Devices with a Touchscreen	744
19.8.4	JavaScript Events That Occur in Connection with the Keyboard	744
19.8.5	JavaScript Events for HTML Forms	745
19.8.6	JavaScript Events for the Web APIs	745
19.9	More Information about Events with the “event” Object	745
19.10	Suppressing the Default Action of Events	748
19.11	The Event Flow (Event Propagation)	749
19.11.1	More about the Bubbling Phase	750
19.11.2	Canceling Bubbling via the “stopPropagation()” Method	751
19.11.3	Intervening in the Event Flow during the Capturing Phase	752
19.11.4	Additional Information on the Capturing and Bubbling Phases	753
19.12	Adding, Changing, and Removing HTML Elements	754
19.12.1	Creating and Adding a New HTML Element and Content	755
19.12.2	Targeting HTML Elements Even More Exactly in the DOM Tree	756
19.12.3	Adding a New HTML Element Even More Targeted to the DOM Tree	760
19.12.4	Deleting an Existing HTML Element from the DOM Tree	762
19.12.5	Replacing an HTML Element in the DOM Tree with Another One	763
19.12.6	Cloning a Node or Entire Fragments of the DOM Tree	764
19.12.7	Different Methods to Manipulate the HTML Attributes	765
19.12.8	The <template> HTML Tag	768
19.13	HTML Forms and JavaScript	770
19.13.1	Reading Text Input Fields with JavaScript	771
19.13.2	Reading Selection Lists with JavaScript	772
19.13.3	Reading Radio Buttons and Checkboxes with JavaScript	773
19.13.4	Intercepting Buttons with JavaScript	775
19.13.5	Controlling the Progress Indicator <progress> with JavaScript	776
19.14	Summary	777
 20 An Introduction to Ajax		779
20.1	An Introduction to Ajax Programming	779
20.1.1	A Simple Ajax Example in Execution	781
20.1.2	Creating the “XMLHttpRequest” Object	783
20.1.3	Making a Request to the Server	783
20.1.4	Sending Data	784
20.1.5	Determining the Status of the “XMLHttpRequest” Object	785
20.1.6	Processing the Response from the Server	787

20.1.7	The Ajax Example during Execution	787
20.1.8	A More Complex Ajax Example with XML and DOM	788
20.1.9	The JSON Data Format with Ajax	793
20.2	Summary	797
	The Author	799
	Index	801

Index

::first-letter (pseudo-element)	330
::first-line (pseudo-element)	330
::selection (pseudo-element)	330
:active (pseudo-class)	319, 588
:any-link (pseudo-class)	318
:blank (pseudo-class)	324
:empty (pseudo-class)	323
:first-child (pseudo-class)	324
:first-of-type (pseudo-class)	326
:focus (pseudo-class)	319
:hover (pseudo-class)	319, 320, 588
:invalid (pseudo-class)	255, 261
:lang() (pseudo-class)	329
:last-child (pseudo-class)	324
:last-of-type (pseudo-class)	326
:link (pseudo-class)	318
:matches() (pseudo-class)	329
:not() (pseudo-class)	329
:nth-child() (pseudo-class)	324
:nth-last-child() (pseudo-class)	324
:nth-last-of-type() (pseudo-class)	326
:nth-of-type() (pseudo-class)	326
:only-of-type (pseudo-class)	326
:placeholder-shown (pseudo-class)	319
:required (pseudo-class)	262
:root (pseudo-class)	323
:target (pseudo-class)	321
:valid (pseudo-class)	255, 261
:visited (pseudo-class)	318
!important	355
@font-face	526
@import (CSS rule)	293, 462, 598
@media (CSS rule)	295, 460, 462
@supports()	441, 596
@viewport	468
>	161
<	161
 	115, 166
"	161
­	115
#anchortname	188
<!-- string	65
<noscript>	645
, list-style-type	558
<th>email address</th>	183, 255
62.5% trick	543

A

a (tag)	177
<i>#anchortname</i>	188
<i>download</i>	186, 192
<i>href</i>	178, 192
<i>href=mailto</i>	183
<i>hreflang</i>	192
<i>media</i>	192
<i>phone number</i>	191
<i>rel</i>	193
<i>Skype</i>	191
<i>target</i>	182, 193
<i>title</i>	186
<i>type</i>	186, 193
abbr (tag)	140
<i>title</i>	140
accesskey	265
addEventListener()	738
address (tag)	110
Adobe Brackets	45
Ajax	779
<i>callback function</i>	785
<i>determining the status</i>	785
<i>DOM</i>	788
<i>example</i>	781
<i>HTTP request</i>	783
<i>HTTP response</i>	787
<i>JSON</i>	793
<i>onreadystatechange</i>	785, 790
<i>open()</i>	784
<i>readyState</i>	785
<i>responseText</i>	787
<i>responseXML</i>	787, 792
<i>send()</i>	784
<i>status</i>	787
<i>XMLHttpRequest object</i>	783
align-content	448
align-items	511
align-self	449, 511
all	353
Anchor	188
and (media query)	463
Angular dimensions	372
any-hover	465
any-pointer	465
appendChild()	755, 760
Application server	41

area (tag)	205	bdi (tag)	144
alt	209	bdo (tag)	144
coords	207, 209	dir	144
download	209	Blisk	598
href	208, 209	blockquote (tag)	117
hreflang	209	cite	117
media	209	Blog	33
rel	209	body (tag)	68, 96
shape	207, 209	Boolean	709
target	209	Boolean data type, JavaScript	666
type	209	Border	395
Array	677	decorative border	396
Array literal notation	689	border	379
article (tag)	98, 135	border (tag)	393
ASCII encoding	158	border-bottom	379
aside (tag)	99, 135	border-box	472
aspect-ratio	464	border-collapse	568
audio (tag)	229	border-color feature	394
autoplay	230	border-image (tag)	396
controls	230	border-image-slice (tag)	396
loop	230	border-image-source (tag)	396
muted	231	border-image-width (tag)	396
preload	231	border-left	379
src	231	border-radius	411
type	230, 231	border-bottom-left-radius	411
Author stylesheet	355	border-top-left-radius	411
Autocompletion	258	border-top-right-radius	411
Automatic redirection	90	border-right	379
B			
b (tag)	146	border-spacing	569
background	402	border-style feature	394
background-attachment	398, 401	border-top	379
background-color	397, 398	border-width feature	394
background-image	398, 498	Box model	375
background-position	398, 401	alternate	386
background-repeat	398, 400	box-sizing: border-box	386
background-size	403, 498	classic	376
linear-gradient()	406	box-shadow	409
radial-gradient()	408	box-sizing	386, 472
repeating-linear-gradient()	407	border-box	388
repeating-radial-gradient()	408	content-box	388
Background color	397	br (tag)	113
Background graphic	399	break, JavaScript	675
fixing	401	Breakpoint	471
positioning	401	Browser stylesheet	355
tiling	399	Bullet point	122
Background image	397	button (tag)	247
base (tag)	76	C	
href	76, 77	calc()	516, 541
target	77	Camel case	734
		canvas (tag)	221

Capital letters	544	const	655
caption (tag)	175	Content area	376
caption-side	571	continue, JavaScript	675
Cascade	291	Corporate website	32
Cascading	345, 354	createElement()	755
Cascading Style Sheets → CSS		createTextNode()	755
Centimeters	364	CSS	42, 279
Central stylesheet	598	<i>alternate stylesheet</i>	291
Character encoding	86, 157	<i>cascade</i>	291
Character entity	160	<i>cascading</i>	354
charset	159	<i>code formatting</i>	285
Chrome	597	<i>commenting code</i>	284
circle (SVG)	218	<i>compression</i>	600
cite, q (tag)	147	<i>inheritance</i>	345
cite (tag)	140	<i>integrating in HTML</i>	285
<i>title</i>	140	<i>in web browser</i>	295
Class	704	<i>level 1 (CSS 1)</i>	280
<i>class selector</i>	302	<i>level 2 (CSS 2)</i>	280
clear	487	<i>level 3 (CSS3)</i>	280
<i>both</i>	439	<i>manipulating</i>	733
<i>left</i>	439	<i>media query</i>	295
<i>none</i>	439	<i>rule</i>	282, 283, 289
<i>right</i>	439	<i>selectors</i>	283, 297
cloneNode()	764	<i>style attribute</i>	286
code (tag)	141	CSS feature	
col (tag)	172	<i>!important</i>	355
colgroup (tag)	172	<i>angular dimensions</i>	372
Collapsing margins	381	<i>background</i>	397
Color	366	<i>background-repeat</i>	400
<i>hexadecimal notation</i>	367	<i>bottom</i>	420
<i>HSL mixture</i>	370	<i>color details</i>	366
<i>named</i>	366	<i>content</i>	330
<i>selecting</i>	251	<i>default value</i>	349
<i>selection dialog</i>	251	<i>forcing inheritance</i>	350
color	465	<i>height</i>	376
Color detail	366	<i>inheritance</i>	345
<i>named colors</i>	366	<i>keyword (value)</i>	365
<i>RGB mixture</i>	368	<i>left</i>	420
<i>transparency</i>	369	<i>margin</i>	379
color-index	465	<i>named colors</i>	366
ColorZilla	370	<i>opacity</i>	405
column-count	555	<i>order</i>	454
column-gap	555	<i>position</i>	417
columns	556	<i>restoring the default value</i>	352
column-width	556	<i>right</i>	420
Combinator	298, 332	<i>short notation</i>	373
Comment	65	<i>string (value)</i>	365
CSS	284	<i>top</i>	420
Comparison operators, JavaScript	667	<i>unit of measurement</i>	363
Conditional statements, JavaScript	665	<i>width</i>	376
Console, JavaScript	647	<i>z-index</i>	431
console object	647		

CSS preprocessor	605, 606	Designing a border	393
<i>installing</i>	607	details (tag)	175, 275
<i>online</i>	607	<i>open</i>	275
CSS pseudo-class	318	dfn (tag)	143
<i>:active</i>	319	dialog (tag)	276
<i>:any-link</i>	318	Directory name	73
<i>:blank</i>	324	Directory structure	73
<i>:checked</i>	320	display	495, 513
<i>:disabled</i>	320	<i>block</i>	513, 585
<i>:empty</i>	323	<i>flex;</i>	443
<i>:enabled</i>	320	<i>grid</i>	501
<i>:first-child</i>	324	<i>inline</i>	513
<i>:first-of-type</i>	326	<i>inline-block</i>	513, 585
<i>:focus</i>	319	<i>none</i>	515
<i>:hover</i>	319, 567	<i>none (CSS element)</i>	459
<i>:invalid</i>	255, 256, 261	div (tag)	118, 132, 583
<i>:lang()</i>	329	dl (tag)	128
<i>:last-child</i>	324	doctype (tag)	66
<i>:last-of-type</i>	326	document.body	755
<i>:link</i>	318	document.documentElement	755
<i>:matches()</i>	329	document object	717
<i>:not()</i>	329	Document Object Model → DOM	
<i>:nth-child()</i>	324	Document outline	105, 107
<i>:nth-last-child()</i>	324	DOM	715
<i>:nth-last-of-type()</i>	326	<i>Ajax</i>	788
<i>:nth-of-type()</i>	326	<i>document object</i>	717
<i>:only-child</i>	324, 326	DOM functions, setAttribute()	730
<i>:placeholder-shown</i>	319	DOM inspector	58, 61, 716
<i>:required</i>	262	DOM manipulation	715, 754
<i>:root</i>	323	DOM method	718
<i>:target</i>	321	DOM object collection	727
<i>:valid</i>	255, 256, 261	<i>baseURI</i>	729
<i>:visited</i>	318	<i>body</i>	729
CSS pseudo-element	330	<i>cookie</i>	729
<i>: </i>	330	<i>doctype</i>	729
CSS reset	600	<i>documentElement</i>	729
<i>normalization</i>	602	<i>documentURI</i>	729
CSS web browser test	593	<i>domain</i>	729
		<i>domConfig</i>	729
		<i>embeds</i>	729
		<i>forms</i>	729
		<i>head</i>	729
		<i>images</i>	729
		<i>implementation</i>	729
		<i>inputEncoding</i>	729
		<i>lastModified</i>	729
		<i>links</i>	729
		<i>readyState</i>	729
		<i>referrer</i>	729
		<i>scripts</i>	730
		<i>title</i>	730
		<i>URL</i>	730

D

Database	41
datalist (tag)	259
Data type, JavaScript	657
DCOMContentLoaded	741
dd (tag)	128
Declaration	283
<i>components</i>	284
Decrement operator, JavaScript	672
default, track (tag)	226
Degree	372
del (tag)	150

DOM property	718, 756	firstChild	756
<i>childNodes</i>	756	flex	451
<i>firstChild</i>	792	flex-basis	451
<i>nodeValue</i>	792	Flexbox	443, 562
DOM tree	58, 716	<i>order</i>	454
do-while loop, JavaScript	674	flex-direction	
Download link	186	<i>column</i>	444
dt (tag)	128	<i>column-reverse</i>	444
Dynamic website	39	<i>row</i>	443
		<i>row-reverse</i>	444
		flex-flow	446
		flex-grow	451
		flex-shrink	451
		flex-wrap	445
		float	434
		<i>inherit</i>	436
		<i>left</i>	435
		<i>none</i>	436
		<i>right</i>	435
		flow-root, display	440
		font	546
		Font Awesome	533
		Font class	522
		font-family	522
		<i>web fonts</i>	527
		Font formatting	546
		Fonts	522
		<i>royalty-free</i>	532
		<i>web font</i>	526
		Font size	536
		<i>em</i>	537
		<i>keyword</i>	537
		<i>pixels</i>	540
		<i>points</i>	540
		<i>relative (em)</i>	537
		<i>rem</i>	539
		font-size	536
		Font stack	522
		font-stretch	548
		Font style	
		<i>bold</i>	543
		<i>italic</i>	543
		font-style	543
		font-variant	544
		font-weight	543
		footer (tag)	108, 135
		for loop, JavaScript	673
		Form	237
		<i>autocomplete</i>	258
		<i>button</i>	247
		<i>checkbox</i>	245
		<i>color selection dialog</i>	251

E

ECMAScript	639
E-commerce website	35
ellipse (SVG)	218
em (tag)	145
embed (tag)	228, 232
empty-cells	570
em quad	364
Event	735
Event handler	736
<i>addEventListener()</i>	738
Event object	746
<i>altKey</i>	747
<i>bubbles</i>	747
<i>button</i>	747
<i>cancelable</i>	747
<i>clientX</i>	747
<i>clientY</i>	747
<i>ctrlKey</i>	747
<i>currentTarget</i>	747
<i>keyCode</i>	748
<i>metaKey</i>	748
<i>preventDefault()</i>	748
<i>screenX</i>	748
<i>screenY</i>	748
<i>shiftKey</i>	747
<i>target</i>	748
<i>type</i>	748
Event propagation	749

F

false, JavaScript	660
Favicon	213
favicon.ico	214
Feature query	596
fieldset (tag)	266, 583, 588
figcaption (tag)	121, 176, 202
figure (tag)	121, 176, 202
File name	72
Firefox	597

Form (Cont.)	
<i>date input field</i>	252
<i>defining a space</i>	238
<i>disabling elements</i>	264
<i>dropdown list</i>	242
<i>email input field</i>	255
<i>entering date and time</i>	253
<i>error during input</i>	261
<i>file upload</i>	246
<i>grouping element</i>	266
<i>hidden input field</i>	248
<i>input fields</i>	250
<i>keyboard shortcut</i>	265
<i>mailer</i>	269
<i>month input field</i>	254
<i>multiline text input field</i>	241
<i>multiple submit buttons</i>	249
<i>number input field</i>	256
<i>password input field</i>	240
<i>phone number input field</i>	256
<i>PHP</i>	268
<i>radio buttons</i>	244
<i>read only</i>	265
<i>regular expressions</i>	260
<i>search input field</i>	254
<i>selection list</i>	242
<i>setting the input focus</i>	258
<i>slider</i>	256
<i>tab sequence</i>	265
<i>text input field</i>	240
<i>text label</i>	245
<i>time input field</i>	252
<i>URL input field</i>	255
<i>using placeholders</i>	260
<i>week input field</i>	254
form (tag)	238, 248
<i>accept-charset</i>	239
<i>action</i>	238, 269, 271
<i>enctype</i>	239
<i>id</i>	248
<i>method</i>	238
<i>method=</i>	271
<i>target</i>	239
Forms, JavaScript	770
fr	502
Function	677
G	
g (SVG)	219
GDPR consent	272
General Data Protection Regulation	
(GDPR)	532
getAttribute()	765
getElementById()	719
getElementsByClassName()	723
getElementsByName()	724
getElementsByTagName()	720
GET method	271
Google Fonts	529
Gradian	372
Gradient	406
Graphic	
<i>embedding</i>	196
<i>link-sensitive</i>	204
grid	501
grid (layout)	501
grid-area	507
grid-column	506
grid-column-end	504
grid-column-gap	511
grid-column-start	504
grid-gap	511
grid-row	506
grid-row-end	504
grid-row-gap	511
grid-row-start	504
grid-template-columns	502
grid-template-rows	502
Grouping columns	172
H	
h1 (tag)	104
h2 (tag)	104
h3 (tag)	104
h4 (tag)	104
h5 (tag)	104
h6 (tag)	104
hasAttribute()	766
hasChildNodes	757
head (tag)	67, 69
header (tag)	108, 135
Heading	104
Height	378
height	464, 571
hidden, main (tag)	121
hover	465
hr (tag)	116
href, area (tag)	208
hsl()	370
hsla()	370

HTML	
<i>adding CSS</i>	285
<i>head data</i>	67
<i>input fields</i>	239
<i>markup language</i>	41
<i>page elements</i>	57
<i>validating</i>	49
html (tag)	67
HTML5 web browser test	594
HTML attribute	64
<i>datetime</i>	151
<i>height</i>	215
<i>id</i>	221
<i>manipulating</i>	732, 765
<i>meta element</i>	92
<i>table elements</i>	169
<i>type</i>	232
<i>width</i>	215
HTML document	638
<i>body</i>	96
<i>framework</i>	65
<i>in browser</i>	56
<i>structure</i>	55
HTML element	59
<iframe>	233
<source>	229
<i>head</i>	69
<i>incorrect nesting</i>	61
<i>interactive</i>	275
<i>nesting</i>	60
<i>omitting tag</i>	62
<i>structuring pages</i>	95
<i>structuring text</i>	111
HTML form	237
HTML tag	59
HTTP request	270
<i>method</i>	239
Hyperlink	163, 177
Hyphen ­	115
I	
i (tag)	146
Icon	213, 532
Icon font	532
ID selector	305
if branch, JavaScript	668
iframe (tag)	228
<i>sandbox</i>	234
<i>seamless</i>	235
Image	
<i>embedding</i>	196
Image (Cont.)	
<i>hiding</i>	495
<i>labeling</i>	202
<i>responsive</i>	489
<i>scaling</i>	201
<i>scaling with CSS</i>	571
Image map	204
img (tag)	196
<i>alt</i>	196, 197, 204
<i>height</i>	200, 204
<i>hiding</i>	495
<i>ismap</i>	204
<i>making responsive</i>	489
<i>name</i>	204
<i>scaling with CSS</i>	571
<i>src</i>	196, 204
<i>src (SVG)</i>	215
<i>SVG</i>	215
<i>title</i>	198
<i>usemap</i>	204
<i>width</i>	200, 201, 204
Inches	364
Increment operator, JavaScript	672
inherit	350
Inheritance	345
CSS	345
<i>inherit</i>	350
initial	352
initial-scale	468
Inline style	286
innerHTML	730
input (tag)	
<i>accept</i>	246
<i>accesskey</i>	265
<i>autocomplete</i>	258
<i>autofocus</i>	258
<i>checked</i>	244
<i>disabled</i>	264
<i>enctype</i>	246
<i>for</i>	245
<i>formation</i>	249
<i>formmethod</i>	249
<i>formnovalid</i>	250
<i>formtarget</i>	250
<i>input type=</i>	240, 247
<i>JavaScript</i>	771, 773
<i>list</i>	259
<i>max</i>	259
<i>maxlength</i>	240
<i>min</i>	259
<i>multiple</i>	255, 260
<i>name</i>	240, 244, 246, 248

input (tag) (Cont.)	
novalidate	263
pattern	256, 260
placeholder	260
readonly	265
required	255, 261, 262
size	240
step	259
tabindex	265
type=	246, 248, 251–256
type=checkbox	245
type=radio	244
value	240, 244, 246, 248, 251, 771
ins (tag)	150
insertBefore()	760
ISO-8859-1	158
J	
JavaScript	
--	672
++	672
<noscript>	645
=== operator	710
alert()	646
arguments object	681
arithmetic operator	663
array	689
array (multidimensional)	691
array functions	686
bool	660
Boolean	709
break	675
canvas (tag)	221
class	704
class syntax	704
comments	652
comparison operators	667
conditional statement	665
confirm()	646
console	647
console.log()	647
const	655
constant	655
constructor function	703
continue	675
converting data types	662
customizing the load behavior	645
data type	657
Date	711
Date object	691
decrement operator	672

JavaScript (Cont.)	
default parameter	681
defining functions	678
disabled	83
do-while loop	674
false	660, 666
for ... in (loop)	693
for ... of (loop)	693
for loop	673
forms	770
Function	711
function	677, 678
function expression	679
function parameter	680
get	706
getter	705
if branch	668
increment operator	672
indexOf()	699
input dialog	646
integration in HTML	641
isNaN()	663
length	700
let	653, 683
logical operator	669
loop	672
Map object	712
Math	711
Math object	665
multiple branching	670
null	661
number	709
numbers	657
object	702, 709
object-oriented programming	702
output	646
parseFloat()	663
parseInt()	663
pop()	694
position	644
prompt()	646
push()	694
queue	696
regular expression	701
rest parameter	682
return statement	683
return value (function)	683
scope (variables)	683
selection operator	669
set	706
Set object	712
setter	705

JavaScript (Cont.)	
shift()	695
sort()	697
splice()	696
stack	695
standard dialog	646
strict mode	656
string	658, 700, 709
substr()	701
substring()	701
switch	670
this	703, 708
traversing arrays	693
true	660, 666
typeof	657
undefined	661
unshift()	695
use strict	656
var	653, 683
variables	652
while loop	674
within HTML	643
JavaScript engine	640
JavaScript event	735
blur	745
bubbling phase	749
capturing phase	749
change	745
click	742
dblclick	742
default action	748
DOMContentLoaded	741
error	740
event handler	736
event propagation	749
focus	745
keydown	744
keypress	745
keyup	745
load	740
mousedown	742
mousemove	742
mouseout	742
mouseover	742
mouseup	742
onclick	773
onkeyup	789
onload	741, 783
onunload	741
preventing the default action	748
properties	745
reset	745

JavaScript event (Cont.)	
resize	740
scroll	741
select	745
submit	745
touchcancel	744
touchend	744
touchmove	744
touchstart	744
unload	741
JavaScript objects	
Date	711
Function	711
Map	712
Math	711
Set	712
JSON	793
JSON.parse()	795
Jump marker	188
justify-content	447
justify-items	511
justify-self	511
K	
kbd (tag)	
142	
L	
label (tag)	
245, 583	
Landing page	36
lang (attribute)	67
lastChild	757
legend (tag)	266
let	653
letter-spacing	547
li (tag)	122, 123
line (SVG)	218
linear-gradient()	406
Line break	
 	113
<wbr>	114
preventing	116
line-height	545
Line spacing	545
Link	177
email	183
insert	178
media	458
text	177
link (tag)	78, 288
href	80, 289

link (tag) (Cont.)
 hreflang 80
 media 80, 293
 rel 80
 rel=\ 213, 289
 size 81
 sizes 213
 type 81
Link-sensitive graphics 204, 209
List
 designing 557
 graphics as bullets 559
List display
 changing the numbering 124
 description list 128
 nesting 125
 numbered 123
 ordered 123
 reversing the numbering 124
 unordered 122
list-style 560
list-style-image 559
list-style-position 560
list-style-type 557
log(), JavaScript 647
Logical operator, JavaScript 669
Loop, JavaScript 672

M

main (tag) 120
 hidden 121
Main content <main> 120
map (tag) 204
margin (tag) 379
margin-bottom (tag) 379
margin-left (tag) 379
margin-right (tag) 379
margin-top (tag) 379
mark (tag) 146
Marker 41
Masking HTML characters 161
math (tag) 220
MathML 219
max-aspect-ratio 464
max-color 465
max-color-index 465
max-height 464
max-monochrome 465
max-resolution 465
max-width 464
Media query 457, 461

Menu bar 561
meta (tag) 85
 charset 92, 157
 charset=\ 86
 content 85, 86, 92
 http-equiv 86, 92
 http-equiv=\ 86
 name 85, 93
 name=\ 88, 89
Metadata 42, 85, 88
meta viewport 87, 467
meter (tag) 268
Microblogging 34
Microsite 36
Microsoft Editor 44
Millimeters 364
MIME type 186
min-aspect-ratio 464
min-color 465
min-color-index 465
min-height 464
min-monochrome 465
min-resolution 465
min-width 464
Model border-box 387
monochrome 465
Multiple branching, JavaScript 670
MySQL 34

N

Named colors 366
Named entity 115, 161
Naming convention 72
nav (tag) 102, 135
Navigation, flexbox 562
Navigation bar 561
new 703
nextSibling 757
nodeName 757
nodeType 757
nodeValue 757
Normalization (CSS) 602
normalize.css 602
noscript (tag) 83
not (media query) 464
null, JavaScript 661
Number 709
Numeric entity 160

O

Object 677, 709
object (tag) 228, 232
ol (tag) 123, 557
 reversed 124
 start 124
 value 125
onblur 771
onchange 772
Online magazine 33
Online store 35
only (media query) 463
onsubmit 775
optgroup (tag) 243
 label 243
option (tag) 242
 JavaScript 772
 selected 243
 value 243, 772
orientation 464
Outline 105, 107
output (tag) 256

P

p (tag) 112
padding 378
padding-bottom 379
padding-left 379
padding-right 379
padding-top 378
Paragraph text <p> 112
parentNode 756
path (SVG) 217
Percent 365
PHP 34
PHP form mailer 269
Pica 364
picture (tag) 210, 496
Pixel 364
Placeholder 260
Plain text format 41
Playing an audio file 229
Playing MP3 229
Point 364
pointer 465
polygon (SVG) 218
Portfolio website 34
position 417
 absolute 422
 fixed 426

position (Cont.)

relative 421
 static (tag) 418
 sticky 429
Positioning 417
 absolute 422
 fixed 426
 float 434
 relative 421
 static 418, 419
POST method 271
pre (tag) 141
Preprocessor 605
preventDefault() 748
previousSibling 757
progress (tag) 267
 JavaScript 776
Protocol 74

Q

q (tag) 147
 cite 147
Query 40
querySelector() 725
querySelectorAll() 725
Query string 271

R

radial-gradient() 408
Radian 372
radius, border-bottom-right-radius 411
reCAPTCHA 185
Recommendation 414
Recordset 40
rect (SVG) 218
Reference text 177
Referencing 73
Regular expression 260
Relative URL 179
rem 539
removeAttribute() 766
removeChild() 762
removeEventListener() 740
repeat() 503
repeating-linear-gradient() 407
repeating-radial-gradient() 408
replaceChild() 763
Request 38
resolution 465
Response 38

Index		Index	
Responsive web design	458	Spam	184
rgb()	368	span (tag)	156
rgba()	405	Specificity	357
RGB mixture	368	Stacking	431
<i>transparency</i>	369	Standalone tag	63
Root directory	75	Static website	38
Root em	365	stopPropation()	751
rotate()	576	Strikethrough	552
Round angle	372	String	700, 709
Round corners	411	<i>JavaScript</i>	658
rp (tag)	155	strong (tag)	145
rt (tag)	155	Style	733
ruby (tag)	155	style (HTML attribute)	286
Ruby annotation	155	style (tag)	81, 287
S		<i>media</i>	82
s (tag)	148	<i>title</i>	291
Safari	597	<i>type</i>	82
samp (tag)	142	sub (tag)	151
sanitize.css	603	Sublime Text	45
Sass	605	submit	775
@content	623	Subtitles for video and audio	225
@each	628	summary (tag)	175, 275
@else	631	sup (tag)	151
@extend	618	svg (tag)	216
@for	630	<i>circle</i>	218
@function	632	<i>ellipse</i>	218
@import	633	<i>line</i>	218
@include	615	<i>path</i>	217
@media	621	<i>polygon</i>	218
@mixin	615	<i>rect</i>	218
@return	632	<i>text</i>	219
@while	630	SVG element, <polyline .../>	218
& (ampersand character)	627	SVG format	214
<i>adjusting brightness</i>	625	<i>img (tag)</i>	215
<i>color</i>	625	<i>svg (tag)</i>	216
<i>comments</i>	634	SVG tag	217
<i>control structure</i>	628	switch, JavaScript	670
<i>function</i>	632	T	
<i>installing</i>	607	tabindex	265
<i>media query</i>	621	Table	163
<i>mixins</i>	615	<i>caption</i>	571
<i>nesting</i>	613	<i>cell</i>	166
<i>operator</i>	624	<i>fixed width</i>	566
<i>property nesting</i>	614	<i>labeling</i>	174
<i>selector nesting</i>	613	<i>structuring</i>	169
<i>variable</i>	611	<i>structuring data</i>	163
<i>Visual Studio Code</i>	608	table (tag)	164
scale()	575	<i>border</i>	169
scope, th (tag)	168	table-layout	567
script (tag)	82	tables	566
<i>async</i>	84, 645	Target anchor	188
<i>charset</i>	84	tbody (tag)	169
<i>defer</i>	84, 645	td (tag)	164
<i>JavaScript</i>	641	<i>colspan</i>	166
<i>src</i>	84	<i>rowspan</i>	166
<i>type</i>	84	template (tag)	768
SCSS → Sass		Template string	660
Search engine optimization (SEO)	72	Test	591
section (tag)	97, 135	Text	
Section element	96	<i>design</i>	521
select (tag)	242	<i>indenting</i>	551
<i>JavaScript</i>	772	<i>subscript <sub></i>	151
<i>multiple</i>	243	<i>superscript <sup></i>	151
<i>name</i>	243	text (SVG)	219
Selection operator, JavaScript	669	text-align	548
Selector	283, 297	Text alignment	548
<i>adjacent sibling combinator</i>	337	<i>vertical</i>	550
<i>attribute selector (attribute value)</i>	312	textarea (tag)	241, 773
<i>attribute selector (partial value)</i>	315	<i>cols</i>	242
<i>attribute selector (presence)</i>	310	<i>maxlength</i>	242
<i>child combinator</i>	335	<i>name</i>	242
<i>class selector</i>	302	<i>rows</i>	242
<i>combinator</i>	332	<i>wrap</i>	242
<i>descendant combinator</i>	334	textContent	728
<i>general sibling combinator</i>	338	text-decoration	552
<i>grouping</i>	301	TextEdit	44
<i>ID selector</i>	305	text-indent	551
<i>negation pseudo-class</i>	329	Text markup	138
<i>pattern</i>	298	text-shadow	554
<i>pseudo-class</i>	318	text-transform	544, 553
<i>structural pseudo-class</i>	322	Text underline	151
<i>type selector</i>	299	tfoot (tag)	169
<i>universal selector</i>	308	th (tag)	165
<i>user interface pseudo-classes</i>	320	<i>scope</i>	168
<i>weighting</i>	357	thead (tag)	169
Semantic HTML	130	time (tag)	151
setAttribute()	765	Time data	372
Shadow	554	title (tag)	70
<i>adding</i>	409	tr (tag)	164
Simple selector	298	track (tag)	225
skew()	577	<i>kind</i>	226
small (tag)	155	<i>label</i>	226
Small caps	544	<i>src</i>	226
source (tag)	210, 496	<i>srclang</i>	226
<i>audio (tag)</i>	230	Transform	
<i>media</i>	211	<i>skewing</i>	577
<i>sizes</i>	211	transform	574
<i>src</i>	223	Transformation	574
<i>srcset</i>	211, 212	<i>moving</i>	577
<i>type</i>	211, 223	<i>rotating</i>	576
Space	115, 238	<i>scaling</i>	575
		Transition	580

transition 580

transition-delay 581

transition-duration 581

transition-property 580

transition-timing-function 581

translate() 577

Transparency 405

true, JavaScript 660

typeof, JavaScript 657

U

u (tag) 148

ul (tag) 122, 557

undefined, JavaScript 661

Underline 552

Unicode 159

Unit of measurement 363

Universal selector

weighting 358

unset 352

URL 73

User stylesheet 355

use strict 656

UTF-8 86, 158, 159

V

Validating

HTML 49

Validation 592

var 653

var (tag) 144

Vector graphic 214

img (tag) 215

svg (tag) 216

vertical-align 550

vh 541

Video

playing 222

YouTube 228

video (tag) 222

autoplay 224

controls 224

height 224

loop 224

muted 224

poster 224

preload 224

src 224

video (tag) (Cont.)

type 225

width 225

Viewport 87, 465

height 365

width 365

Viewport unit 541

visibility 515

Visual Studio Code 45

vw 541

W

wbr (tag) 114

Web app 37

Web browser 46, 592

default stylesheet 355

Web browser prefix 413

Web crawler 85, 88

Web font 526

Web form 237

Weblink 177

Weblog 33

Web page

create 47

Web platform 36

Web presence 32

Web server 40

WebVTT format 227

Weighting 357

while loop, JavaScript 674

Width 378

width 464, 571

viewport 467

Word spacing 547

word-spacing 547

Working draft 414

WYSIWYG editor 45

X

x-height 364

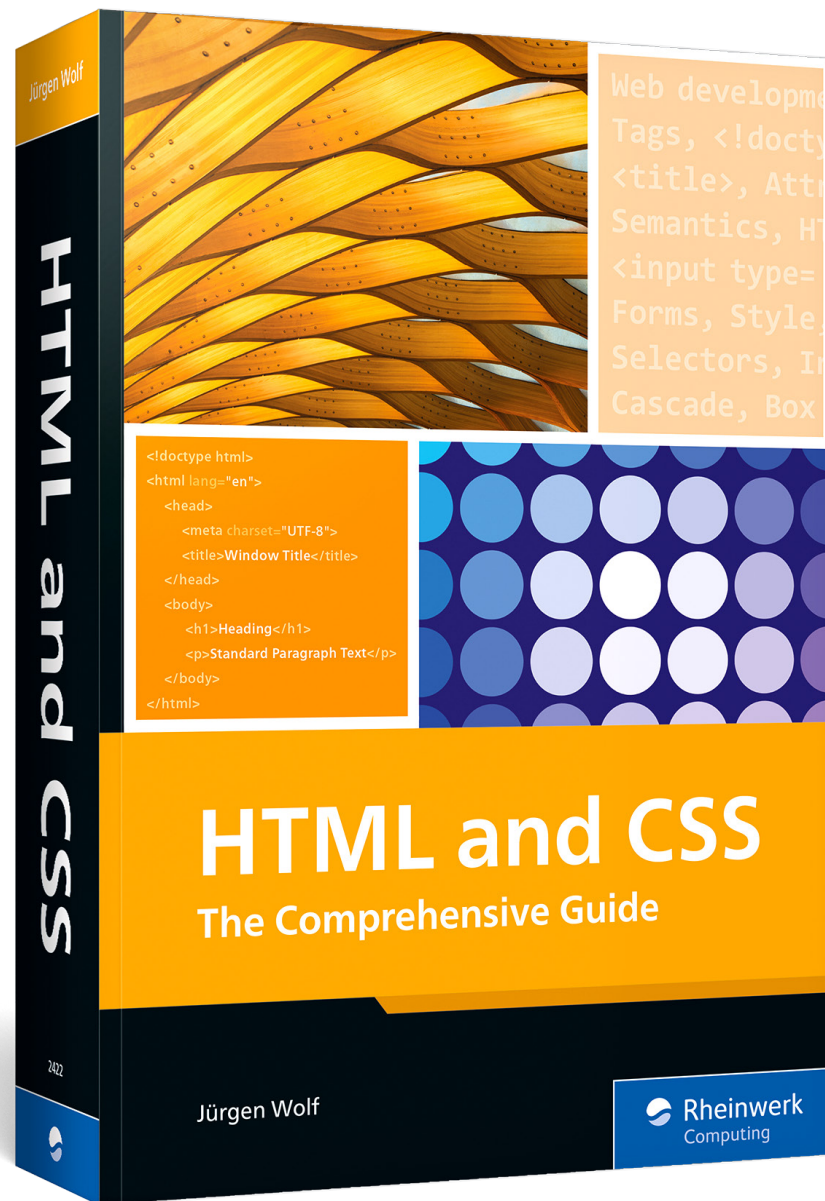
XMLHttpRequest object 783

Y

YouTube 228

Z

z-index 431



Jürgen Wolf is a web and software developer and the author of several seminal works about programming and photography.

Jürgen Wolf

HTML and CSS

The Comprehensive Guide

814 pages | 04/2023 | \$59.95 | ISBN 978-1-4932-2422-7

 www.rheinwerk-computing.com/5695

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.