Reading Sample

This sample chapter covers CSS, a style language for controlling how certain HTML web page elements are visualized in the frontend. It discusses how to use CSS to specify the font in which text should be displayed and its color. Next, it walks through managing the appearance of lists, form elements, and tables. Finally, it covers the different systems that allow you to change the layout of your web page.

“Designing Web Pages with CSS”

Contents

Index

The Authors

Philip Ackermann

Full Stack Web Development

The Comprehensive Guide

740 pages | 08/2023 | $59.95 | ISBN 978-1-4932-2437-1

www.rheinwerk-computing.com/5704

Chapter 3
Designing Web Pages with CSS

You can influence the appearance of web pages by using the Cascading Style Sheets (CSS) style language.

As mentioned earlier, *CSS* is a style language for controlling how certain Hypertext Markup Language (HTML) elements of a web page are visualized in the frontend. For example, you can use CSS to specify the font in which a text should be displayed and its color. You can influence the appearance of lists, form elements, and tables, for example, setting the bullet points in a list or determining the background color of individual table cells. So, you have many options to make HTML, which looks rather dull by default (as shown in the figures in Chapter 2), more attractive and appealing.

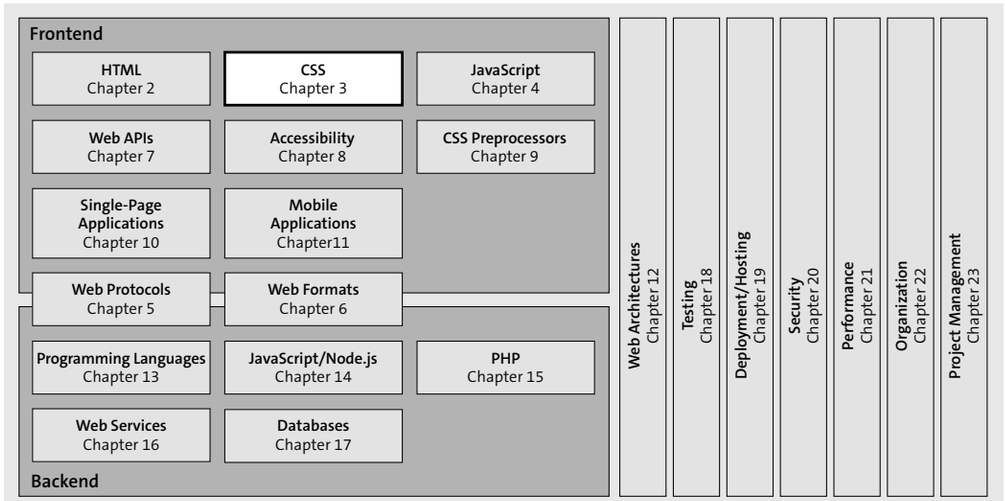

Figure 3.1 CSS, One of Three Important Languages for the Web, Defines the Appearance of a Web Page

3.1 Introduction

In this section, I'll show you how *CSS stylesheets* are structured and how you can include CSS in HTML code. I'll also provide an overview of the most important terms and concepts of the language.

3.1.1 The Principle of CSS

You can define how the content of certain HTML elements should be displayed using *CSS rules*. These rules basically consist of two parts, as shown in Figure 3.2. The *CSS selector* enables you to specify which HTML elements should be subjected to the respective CSS rule. You can use the *CSS declaration* written in curly brackets to specify how exactly these HTML elements should be displayed. Declarations, in turn, consist of a *CSS property* and a *CSS value*, both separated by a colon and ending together with a semicolon.

For example, properties can affect the color, font, dimensions, or border color of an element. Using the value of the property, you can then specify, for example, which color, which font, which width or height, or which frame color should be selected. Each property has certain predefined values that are valid.

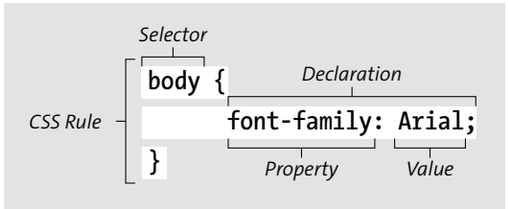

Figure 3.2 The Structure of CSS Rules

In the course of this chapter, I'll show you what types of selectors and what properties are available. First of all, for illustration purposes, let's consider a simple CSS rule.

```
h1 {
  font-family: Arial;
  color: darkblue;
}
```

Listing 3.1 A Simple CSS Rule

This CSS rule states that all first-level headings (defined by the `h1` selector) should use the "Arial" font (defined by the `font-family` property) and be displayed in dark blue (defined by the `color` property).

3.1.2 Including CSS in HTML

In total, three different ways exist for defining CSS rules and including them in an HTML document:

- **External stylesheets (external CSS)**
  In this case, you save the CSS instructions as a separate CSS file (with file extension *.css*) and include this file in the HTML document.

- **Internal stylesheets (internal CSS)**
  In this case, you define the CSS instructions in the header of the HTML document within the `<style>` element.
- **Inline styles (inline CSS)**
  In this case, you specify the CSS instructions directly in an HTML element.

Including External CSS Files (External CSS)

Let's start with the variant that makes the most sense in most cases: the inclusion of separate CSS files (*external CSS*). The reason why this variant is often the most useful is you can cleanly separate the CSS code from the HTML code and thus easily include it in multiple HTML documents. Thus, web projects can have one central CSS file (or a few files) that can then be included in all HTML documents in the project. This approach ensures that each HTML document uses the same CSS instructions and that the presentation of each web page is consistent. This structure also has a considerable advantage with regard to modifications: When you make style changes, you only need to make them once centrally, in the shared CSS files, which will affect all HTML documents, as shown in Figure 3.3.

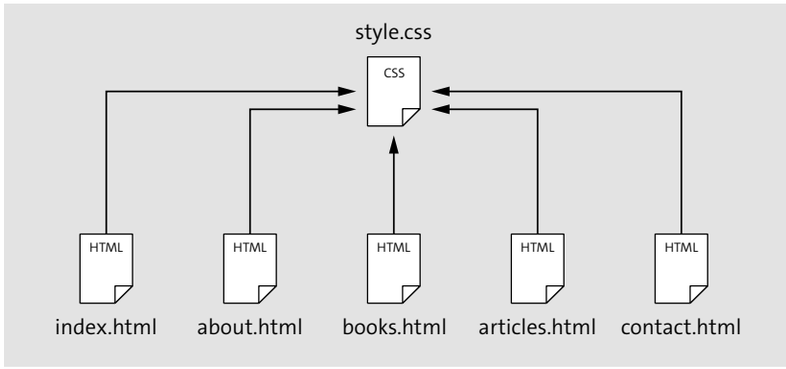

Figure 3.3 Reusing External CSS Files in Different HTML Files

CSS files are ordinary text documents that you can create and edit in any text editor with the *.css* file extension (for example, *styles.css*). Listing 3.2 shows an example of the contents of a simple CSS file that contains a total of three CSS rules.

```
/* File styles.css */
body {
  font-family: Arial;
  background-color: lightblue;
}
```

```
h1 {
  color: darkblue;
}

h2 {
  text-transform: uppercase;
}
```

Listing 3.2 A Simple CSS File

In Listing 3.3, we show you how to include this external CSS file in an HTML document using the `<link>` element. You can specify the URL or the path to the CSS file via the `href` attribute. Since the `<link>` element can basically also be used for including other file types, you can also define the MIME type of CSS via the `type` attribute (see also Chapter 6). In addition, with the `rel` attribute, you can define whether the browser should use the respective CSS file as the primary stylesheet (`stylesheet` value) or ignore it until the user or an application explicitly activates the stylesheet (`alternate stylesheet` value).

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first web page with CSS</title>.
    <link href="css/styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1>My first web page</h1>
    <p>This is a paragraph.</p>
    <h2>This is a subheading</h2>.
    <p>
      Here is another paragraph with <i>italics</i> and
      <b>bold text</b>.
    </p>
    <h2>This is another subheading</h2>
  </body>
</html>
```

Listing 3.3 Including an External CSS File in an HTML Document

When you now load the HTML file, the browser finds the specified CSS file and applies the CSS rules to the corresponding HTML elements. So, the web page should look similar to the one shown in Figure 3.4.

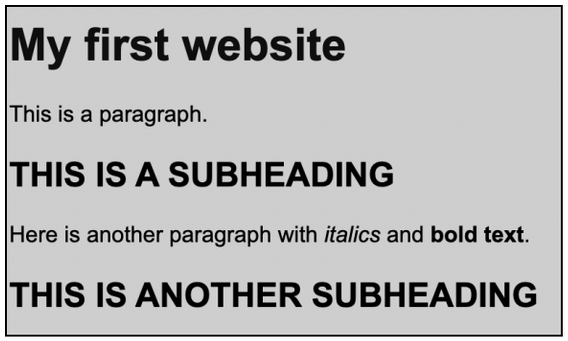

Figure 3.4 HTML with CSS Rules Applied

Defining CSS Instructions in the HTML Document (Internal CSS)

You can also define CSS rules directly within an HTML document (*internal CSS*). However, this option should be an exception. When you "mix" CSS code and HTML code, you cannot reuse the CSS code elsewhere (i.e., in other HTML documents) unless you copy it over manually, which is even less advisable because then keeping changes in sync becomes quite difficult.

To define CSS rules directly in an HTML document, you can simply write them out as text in the `<style>` element, which is usually located in the `<head>` element of the web page. The sample code shown in Listing 3.4 illustrates how you can define CSS rules in this way.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first web page with CSS</title>.
    <style type="text/css">
      body {
        font-family: Arial;
        background-color: lightblue;
      }

      h1 {
        color: darkblue;
      }

      h2 {
        text-transform: uppercase;
      }
    <style>
```

```
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>My first web page</h1>
    <p>This is a paragraph.</p>
    <h2>This is a subheading</h2>.
    <p>
      Here is another paragraph with <i>italics</i> and
      <b>bold text</b>.
    </p>
    <h2>This is another subheading</h2>
  </body>
</html>
```

Listing 3.4 Directly Defining CSS Code Internally in HTML Code

Note

For most examples in this chapter, I define the CSS code directly within the HTML code for didactic reasons. In this way, you can see both the CSS code and the HTML code clearly arranged in a single listing.

Defining CSS Rules as an Attribute (Inline CSS)

Finally, you can also define CSS directly in a single HTML element (*inline CSS*). For this task, simply define the appropriate CSS declarations within the `style` attribute of the HTML element. However, you should consider this option only in exceptional cases since even the reusability of CSS rules within an HTML document is lost. In the next example, you can see this limitation with the CSS declarations for the `<h2>` elements: You must define the CSS declaration `text-transform: uppercase;` separately for each element, which is not at all ideal for reasons of reusability.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My first web page with CSS</title>.
  </head>
  <body style="font-family: Arial; background-color: lightblue;">
    <h1 style="color: darkblue;">My first web page</h1>
    <p>This is a paragraph.</p>
    <h2 style="text-transform: uppercase;">
      This is a subheading
    </h2>
```



```
    <p>
      Here is another paragraph with <i>italics</i> and
      <b>bold text</b>.
    </p>
    <h2 style="text-transform: uppercase;">
     This is another subheading
    </h2>
  </body>
</html>
```

Listing 3.5 Defining CSS Code in HTML Elements

Note

In addition to these ways for including CSS, a fourth option exists but is only available within CSS files or within CSS defined via the `<style>` element. To import additional CSS files, you can use the `@import` rule.

```
@import url(/styles.css);
@import url(/styles/more-styles.css);
```

Listing 3.6 A Simple CSS File

CSS rules can be made even more reusable via imports. For example, you can store the rules for table layouts in a *tables.css* file, the rules for *forms* in a *forms.css* file, and rules for basic layout in a *structure.css* file. You could then access this construction kit of CSS rules relatively flexibly and include different files depending on the project or website.

3.1.3 Selectors

To define which HTML elements are affected by a CSS rule, you can use different types of *selectors*. For example, you can select elements by their tag name, by the `id` attribute, by CSS classes, by attributes, and by the hierarchy in which an element resides. Table 3.1 contains an overview of how to use selectors.

Selector	Description	Code Example	Description of the Code Example
Type selector	Selects all elements that match the specified type or element name.	`input { color: green; }` `h1, h2 { color: green; }`	The `input` selector selects all `<input>` elements, the `h1, h2` selector selects all `<h1>` and all `<h2>` elements.

Table 3.1 Types of CSS Selectors

Selector	Description	Code Example	Description of the Code Example
Class selector	Selects elements based on CSS classes, that is, elements whose value of the `class` attribute corresponds to the value after the dot in the selector.	`.valid { color: green; } a.valid { color: green; }`	The `.valid` selector selects all elements whose `class` attribute has the value `valid`. The `a.valid` selector selects all `<a>` elements whose `class` attribute has the value `valid`.
ID selector	Selects elements whose value of the `id` attribute corresponds to the value behind the hash symbol in the selector.	`#start { color: green; }`	The `#start` selector selects all elements whose `id` attribute has the value `start`. `id` attributes should be unique within a web page and, related to the example, only one element should have the "start" ID. The `p#start` selector selects all `<p>` elements whose `id` attribute has the value `start`.
Universal selector	Selects all elements.	`* { color: green; }`	The * selector selects all elements on a web page.
Attribute selector	Selects elements based on the value of one of their attributes. In this context, the following applies: ■ `[attribute]`: The attribute occurs in the respective element. ■ `[attribute=value]`: The attribute occurs and has exactly the specified value.	`[type="text"]{ color: green; }`	The `[type="text"]` selector selects all text fields (i.e., all `<input>` elements whose `type` attribute has the value `text`).

Table 3.1 Types of CSS Selectors (Cont.)

Selector	Description	Code Example	Description of the Code Example
Attribute selector (Cont.)	■ `[attribute~=value]`: The attribute contains, as its value, a list of which one list element corresponds exactly to `value`. ■ `[attribute|=value]`: The attribute either has exactly the specified `value` value, or the value was hyphenated and starts with `value` followed by a hyphen. ■ `[attribute^=value]`: The attribute has a value that starts with `value`. ■ `[attribute$=value]`: The attribute has a value that ends with `value`. ■ `[attribute*=value]`: The attribute has a value that contains `value` at least once.		
Adjacent sibling selectors	Selects elements that immediately follow another element (called *sibling elements*).	`h1 + p { color: green; }`	The `h1 + p` selector selects all text sections (`<p>` elements) that immediately follow a level-one heading (`<h1>` element).
General sibling selectors	Selects elements that follow another element (but not necessarily immediately).	`h1 ~ p { color: green; }`	The `h1 ~ p` selector selects all text paragraphs (`<p>` elements) that follow a level-one heading (`<h1>` element) at some point.

Table 3.1 Types of CSS Selectors (Cont.)

Selector	Description	Code Example	Description of the Code Example
Child selectors	Selects elements that are direct child elements of the other element defined in the selector.	`body > ul { color: green; }`	The `body > ul` selector selects all unordered lists (`<ul>` elements) that occur directly below the `<body>` element. Nested lists would therefore not be affected by this selector.
Descendant selectors	Selects elements that are descendants of the other element defined in the selector (but not necessarily direct child elements).	`body ul { color: green; }`	The `body ul` selector selects all unordered lists (`<ul>` elements) that occur anywhere below the `<body>` element, including nested lists.

Table 3.1 Types of CSS Selectors (Cont.)

Combining Selectors

Selectors can of course also be combined with each other, making quite complex selection rules possible. For example, you can craft a selector that selects all `<input>` elements whose `class` attribute has the value `important` (`.important` selector), whose `type` attribute has the value `text` (`[type="text"]` selector), and which are immediate child elements of a `<form>` element (`>` selector) with the ID "order-details" (`#order-details` selector).

```
form#order-details > input.important[type="text"]
```

Listing 3.7 Combining Multiple Selector Types

3.1.4 Cascading and Specificity

You can use selectors to specify to which elements a CSS rule should be applied. However, you must understand how CSS rules affect the child elements of elements selected in this way because styles are applied in a *cascading manner*.

In general, the word "cascade" refers to a waterfall that falls over several steps. In terms of CSS, the principle of cascading helps in defining CSS rules. Thus, more general rules can be defined that apply to many elements (which are thus arranged on a higher level in the "waterfall" and "reach" several elements), as well as more specific rules that apply to a few or even only a single element (and which are arranged on a lower level in the

waterfall analogy). Due to this cascading of CSS rules, sometimes of course (intentionally or unintentionally) several CSS rules may apply to one element. The CSS standard defines exactly which CSS rules takes precedence in such cases.

For two *selectors that are the same* and apply to the same element, the second selector takes precedence. In our example shown in Listing 3.8, level-one headings (`<h1>` elements) are displayed in green color (`green`) because the corresponding CSS rule in the CSS code was defined after the CSS rule that colors the text blue. However, note that yellow is used as background color (`yellow`) because this CSS declaration is not "overridden" in the second CSS rule.

```
h1 {
  color: blue;
  background-color: yellow;
}

h1 {
  color: green;
}
```

Listing 3.8 If the Selectors Are the Same, the CSS Rule of the Last Defined Selector Is Used

For two *different selectors*, the selector with the higher *specificity* takes precedence. For example, a concrete selector like `input` is more specific than a general `*` selector, so the former has higher specificity. In the example shown in Listing 3.9, level-one headings(`<h1>` elements) are also rendered in green color (`green`), although the CSS rule that colors the text in blue was defined according to the other CSS rule. The reason for this result is the higher specificity of the selector of the first CSS rule: The `h1` selector is more specific than the general `*` selector.

```
h1 {
  color: green;
}

h1.chapter {
  color: orange;
}

* {
  color: blue;
  background-color: yellow;
}
```

Listing 3.9 If the Selectors Are Different, the CSS Rule of the More Specific Selector Is Used

Calculating the Specificity

The calculation of the *specificity* of a selector is based on three counters (A, B, and C), each of which has the initial value 0. Counter A is incremented by 1 for each ID selector, counter B is incremented by 1 for each occurrence of an attribute or class selector or *pseudo-class* (see box), and finally counter C is incremented by 1 for each occurrence of a type selector or *pseudo-element* (see box). Selectors other than those mentioned, such as the universal selector, are not considered when calculating specificity.

Let's look at some examples where the selectors become more specific from top to bottom.

```
* {}               /* A=0, B=0, C=0, specificity --> 0 0 0 */
h1 {}              /* A=0, B=0, C=1, specificity --> 0 0 1 */
ol li {}           /* A=0, B=0, C=2, specificity --> 0 0 2 */
div:first-child {} /* A=0, B=1, C=1, specificity --> 0 1 1 */
a.tests[href] {}   /* A=0, B=2, C=1, specificity --> 0 2 1 */
#anchor {}         /* A=1, B=0, C=0, specificity --> 1 0 0 */
#chapter p {}      /* A=1, B=0, C=1, specificity --> 1 0 1 */
```

Listing 3.10 Examples of Calculating the Specificity for Better Readability with Empty CSS Rules

Furthermore, CSS rules defined via the `style` attribute—even if they don't have a selector in that sense—are considered more specific.

Pseudo-Classes and Pseudo-Elements

Pseudo-classes select regular elements but only under certain conditions, for example, if their position is relative to siblings or if they are in a certain *state*. Some examples of pseudo-classes include the following (see also Section 3.2.2):

- `:link` denotes links in the HTML code.
- `:visited` denotes already visited links.
- `:hover` denotes links that are currently "touched" by the mouse.
- `:active` denotes links that are currently active.
- `:focus` denotes links that currently have the focus.

Pseudo-elements, on the other hand, address specific *parts of a selected element* and sometimes even *create new elements* that are not specified in the HTML code of the document and can then be edited in a similar way to a regular element. Examples of pseudo-elements include the following:

- `::before` creates an element before the element selected by the selector.
- `::after` creates an element after the element selected by the selector.
- `::first-letter` refers to the first letter of the text in the selected element.

- `::first-line` refers to the first line of text in the selected element.
- `::selection` refers to the part of the text in the selected element that is currently selected by the user.

3.1.5 Inheritance

Some CSS property values set via CSS rules for parent elements are inherited by their child elements. For example, if you specify a color (`color`) and font (`font-family`) for an element, every element in it will also be styled with that color and font unless other color and font values have been explicitly applied to it directly (for example, through other CSS rules). Some properties, on the other hand, such as those for defining borders (`border`), are not inherited. Otherwise, setting a border for a single element would result in the same border being displayed for all child elements, which would be a bit too much of a good thing.

3.2 Applying Colors and Text Formatting

In this section, I'll show you how to use CSS to define the color of elements on a web page and how to format text.

3.2.1 Defining the Text Color and Background Color

To define the color of a text (or the *foreground color* in general), you can use the CSS property `color`. You can define the color value in several different ways:

- RGB values: In this case, you define the value as a composition of red, green, and blue components, where each component is expressed by a number between 0 and 255. For example, the CSS value `rgb(255,255,0)` represents the color yellow.
- Hex values: In this case, you define the value as a 6-digit hexadecimal value, with 2 digits each for the red value, 2 digits for the green value, and 2 digits for the blue value. For example, the color yellow would be represented with the CSS value `#FFFF00`. In addition, a short notation is used for 216 *web colors* (for example, `#FF0` for `#FFFF00` or `#F90` for `#FF9900`).
- Color names: In this case, you specify the color name of the desired color. Over 140 predefined color names are available. For example, you would define the color yellow using the name `yellow`. In addition, exotic color names are available, like `hotpink`, `deepskyblue`, or `lavenderblush`. A detailed overview of the available color names can be found at *https://www.w3.org/wiki/CSS/Properties/color/keywords*.
- RGBA values: Since CSS3, you can also define color values using an RGB value with an additional value for specifying the opacity, called the *alpha value*. This value is

between 0.0 and 1.0 and determines how transparent the color is. For example, you would define a yellow of 50% using the CSS value `rgba(255,255,0,0.5)`.

- HSL values: Also, since CSS3, you can define color values based on *hue*, *saturation*, and *lightness*. A hue is expressed as an angle between 0° and 360°, and saturation and lightness are percentages. For example, you would define the color yellow using the CSS value `hsl(60,100,50)`.
- HSLA values: Similar to the definition of RGBA values, in the case of HSL, since CSS3, you can specify a fourth value for determining opacity. A yellow of 50% in this case would have the CSS value `hsla(60,100,50,0.5)`.

Some examples of defining colors using CSS are shown in Listing 3.11.

```
h1 {
  color: darkblue;
}

h2 {
  color: #ffa500;
}

p {
  color: rgb(169, 169, 169);
}
```

Listing 3.11 Defining Text Colors via Color Names, Hexadecimal Values, and RGB Values

You can set the *background color* of an element using the CSS property `background-color`. For this property, you can use the same values as for the `color` property (i.e., RGB values, hex values, color names, etc.).

```
body {
  background-color: grey;
}

11 {
  background-color: #ffa500;
}

p {
  background-color: rgb(169, 169, 169);
}
```

Listing 3.12 Defining Background Colors via Color Names, Hexadecimal Values, and RGB Values

3.2.2 Designing Texts

Using CSS, in addition to defining text colors, you can influence the basic appearance of text on a web page. For example, you can define *fonts*, adjust *font sizes* or *styles*, and adjust the spacing between words (*word spacing*) or individual letters (*letter spacing*).

Figure 3.5 shows some text formatting options possible using CSS. We'll look at these CSS properties in detail in the following sections.

Figure 3.5 The Result of Text Formatted with CSS

Defining Fonts

First, CSS can define the font for a text. The only requirement for displaying the font correctly when the corresponding web page is called up is that the font must be installed or available on the computer.

The property for specifying the font is called `font-family`, and this property expects the name of the desired font as its value. Optionally, you can specify multiple fonts separated by commas. These fonts then serve as *fallback fonts*: If the font you specified first in this list is not installed on a computer, the browser then has the option of falling back on other fonts specified in the list.

As shown in Listing 3.13, for example, the "Times New Roman" font is first defined for the `body` element. If this font is not available on a computer (which is rare because this default font is available on all computers, but for demonstration purposes let's assume this is the case), then the browser can fall back to the more general "Times" font family and use a font from this family. If neither is available, the "serif" specification says that the browser may use any *serif font*.

```
body {
  font-family: 'Times New Roman', Times, serif;
}

h1, h2 {
  font-family: Arial, sans-serif;
}

ul li {
  font-family: 'Courier New', Courier, monospace;
}

.author {
  font-family: Verdana;
}

.date {
  font-family: 'Courier New', Courier, monospace;
}
```

Listing 3.13 Defining Fonts in CSS

Note

If the name of a font contains spaces, the entire name must be enclosed in quotation marks.

For good style, you should always specify the *general name* of a font as a *fallback font* last when using `font-family`. Possible values in this context include the following:

- `serif`
  Serif font, that is, a font with small hooks on the main strokes of the letters, commonly used in printed text.
- `sans-serif`
  Sans-serif font, that is, a font without small hooks but with straight letter ends. As a rule, sans-serif fonts are more suitable than serif fonts for displaying text on screens because they are easy to read regardless of the screen resolution due to the comparatively less detail.
- `monospace`
  Non-proportional font, that is, a font in which all characters have the same width. This type of font is particularly well suited for displaying source code on a web page. So, for example, if you're writing a blog about web development and want to include code samples, you might use a non-proportional font.

- `cursive`
  Cursive font or *italic font*, that is, a font that looks as handwritten. I would recommend these types of fonts for web page design only in exceptional cases, for example, to set visual accents, but not for the display of continuous text.
- `fantasy`
  Fantasy font, that is, a font that contains decorative elements and, like the cursive/italic font, is more suitable for discreet use and not for displaying continuous text.

Adjusting the Font Size

You can modify the size of a font using the `font-size` property. Several options are available when specifying font size: Among other ways, you can specify font size via a pixel value, a percentage value, or the "em" unit of measurement (which is based on the width of the letter "m").

Pixel values specify the size of a font in screen pixels: A font with the specification 14px ("px" for pixel) is exactly 14 pixels high.

The specification of font size in percentages is based on the standard font size, which is preset in the browser or has been configured by a user through computer settings. By default, most browsers use a font size of 16 pixels. Thus, if you specify a font size for headings as 150%, for example, the font size of the heading will be 24 pixels.

The specification of font size using the "em" unit is also based on the font size. For example, as shown in Listing 3.14, the font size for headings is set to 1.5 times the font size used in each case.

```
body {
  /* font size of 14 pixels */
  font-size: 14px;
}
h1 {
  /* 150% of the regular font size */
  font-size: 150%;
}

h2 {
  /* 1.5 times the width of the letter m */
  font-size: 1.5em;
}
```

Listing 3.14 Adjusting Font Size with CSS

Adjusting the Font Style

The font style of a text, that is, whether the text is to be displayed in italics or bold, can be specified via the `font-style` (italics) or `font-weight` (bold) properties.

The `font-style` property takes one of the following values: The `italic` value provides italic text, and the `oblique` value provides oblique text. (See the box for the differences between "italic" and "oblique.") The `normal` value provides normal text (i.e., neither italic nor oblique).

With the `font-weight` property, you can specify the "weight" of a font, that is, whether the text should be displayed normally (`normal` value) or bold (`bold` value). In addition to these two values, `lighter` and `bolder` are also available, which make the text one level thinner and bolder, respectively, than the text of its parent element. Furthermore, individual gradations can also be defined as numerical values (in hundredths) between 100 and 900. For example, the `normal` value corresponds to the numeric value `400`, while the `bold` value corresponds to the numeric value `700`.

```
.introduction {
  /* italics */
  font-style: italic;

  /* Bold font */
  font-weight: bold;
}
```

Listing 3.15 Adjusting Font Style with CSS

Note

The difference between `italic` and `oblique` is subtle: In the former, the browser should display the text in a font style where the characters of the font have been specially *optimized and designed* for italic printing. If no such special font style is available for a font, the `oblique` value forces the browser to *tilt* the corresponding font. However, the result is usually not as visually appealing as with a real italic font.

Other Font Formatting Options

In addition to formatting font, font size, and font style, CSS offers many other ways to format fonts:

- You can use the `text-transform` property to switch between different *notations*. The `uppercase` value represents the entire text in uppercase, and the `lowercase` value, in lowercase. By using `capitalize`, you can specify that the first letter of each word in a text should be capitalized.
- The `text-decoration` property allows you to *"decorate" text by using lines*, that is, to strike through (`line-through` value) or underline (`underline`), to draw a line across the text (`overline`), and even to make the text blink (`blink`). However, you should avoid the latter if possible, as blinking text on a web page can be annoying. You should also

avoid underlining text because underlined text is usually interpreted by users as a link. Users might be confused if no link opens when they click on the text.

- To influence the *line spacing* of a text, you can use the `line-height` property. The spacing between individual letters (*letter spacing*) can be controlled by using the `letter-spacing` property, while the spacing between individual words (*word spacing*) can be managed via the `word-spacing` property.
- You can manipulate the *horizontal alignment* of text using the `text-align` property. Possible values are `left` (left alignment), `right` (right alignment), `center` (centered alignment), and `justify` (`justified` alignment).
- The *vertical alignment*, on the other hand, is determined by the `vertical-align` property. Text can be aligned `top`, `bottom`, or `middle`.
- The `text-indent` property can be used to *indent* the first line of text.
- The `text-shadow` property can be used to add a *shadow effect* to text.

Formatting Links

Links are a special case of text and are specially marked by the browser by default (often underlined and in blue). You can use *pseudo-classes* to customize the appearance and behavior of links via CSS. The pseudo-class is used in the following selectors, as shown in Listing 3.16:

- `:link`: Allows the formatting of links that have not yet been visited.
- `:visited`: Allows the formatting of links that have already been visited.
- `:hover`: Allows the formatting of links when "touched" by a mouse cursor.
- `:active`: Allows the formatting of links the moment they are clicked on.
- `:focus`: Allows the formatting of links when they receive focus.

```
/* Links */
a:link {
  color: blue;
}

/* Already visited links */
a:visited {
  color: green;
}

/* Links that are currently "touched" with the mouse */
a:hover {
  color: black;
  background-color: orange;
}
```

```
/* Links that are currently being clicked on */
a:active {
  color: red;
}

/* Links that currently have the focus */
a:focus {
  color: orange;
}
```

Listing 3.16 Formatting Links via Pseudo-Classes

Complete Example: Designing Texts

The full code for the web page shown in Figure 3.5 earlier in this section is provided in Listing 3.17. In this example, we've included several CSS properties for formatting texts. Look at the CSS code and the HTML code at your leisure and try to understand which CSS rules apply to which HTML elements. Of course, the best way to explore this code interactively is to download the source code for the listing from the web page for this book and open it in a browser. The developer tools of modern browsers, such as Chrome DevTools, are also useful in this regard. Use these tools to examine exactly which CSS rules are applied to which HTML elements (refer to the practical tip in the info box).

```
<!DOCTYPE html>
<html>
<head>
  <title>Formatting Fonts</title>
  <style type="text/css">
  body {
    font-family: 'Times New Roman', Times, serif;

    /* line height 1.5 times the normal font size */
    line-height: 1.5em;
  }

  h1, h2 {
    font-family: Arial;

    /* character spacing 0.2 times the normal font size */
    letter-spacing: 0.2em;

    /* character spacing 0.3 times the normal font size */
    word-spacing: 0.3em;
  }
```



```
h1 {
  /* 150% of the regular font size */
  font-size: 150%;
}

h2 {
  /* 100% of the normal font size */
  font-size: 100%;

  /* all uppercase letters */
  text-transform: uppercase;

  /* text underlined */
  text-decoration: underline;
}

p {
  /* Justification */
  text-align: justify;
}

ul li {
  font-family: 'Courier New', Courier, monospace;
}

/* Links */
a:link {
  color: blue;
}

/* Already visited links */
a:visited {
  color: green;
}

/* Links that are currently "touched" with the mouse */
a:hover {
  color: black;
  background-color: orange;
}

/* Links that are currently being clicked on */
a:active {
  color: red;
```

```
}

.introduction {
  /* italics */
  font-style: italic;

  /* Bold font */
  font-weight: bold;
}

.author {
  font-family: Verdana;

  /* line height 0.3 times the normal font size */
  line-height: 0.3em;

  /* text right-aligned */
  text-align: right;
}

.date {
  font-family: 'Courier New', Courier, monospace;

  /* line height 0.3 times the normal font size */
  line-height: 0.3em;

  /* text right-aligned */
  text-align: right;
}
</style>
<meta charset="UTF-8">
</head>
<body>
<h1>Blog posts</h1>
<article>
  <h2>New website</h2>
  <p class="introduction">
    Dear readers,
  </p>
  <p>
    From now on you will find the blogs about my books combined on this
website. In this way, you are always kept up to date at a single central point
when it comes to updates about the books mentioned or general information,
tutorials etc. about web and software development.
```



```
  </p>
  <p>
    You can also find news and updates in short microblogging short form on
Twitter:
  </p>
  <p>
    <ul>
      <li>
          <a href="https://twitter.com/cleancoderocker">@cleancoderocker</a>
      </li>
      <li>
          <a href="https://twitter.com/webdevhandbuch">@webdevhandbuch</a>
      </li>
      <li>
          <a href="https://twitter.com/nodejskochbuch">@nodejskochbuch</a>
      </li>
      <li>
          <a href="https://twitter.com/jshandbuch">@jshandbuch</a>
      </li>
      <li>
          <a href="https://twitter.com/jsprofibuch">@jsprofibuch</a>
      </li>
    </ul>
  </p>
  <p>
    Have fun with it!
  </p>
  <p class="author">Philip Ackermann</p>
  <p class="date">May 2020</p>
</article>
</body>
</html>
```

Listing 3.17 Formatting Text with CSS

Practical Tip

In practice, you may no longer have a direct overview of which HTML element is affected by which CSS rule or why an element is assigned a certain property (whether explicitly or through inheritance). In this case, the browser' developer tools, for example, Chrome DevTools, can help. Within the **Elements** section, various features are available. For example, the **Styles** tab lets you view the applicable CSS rules for a selected element, as shown in Figure 3.6.

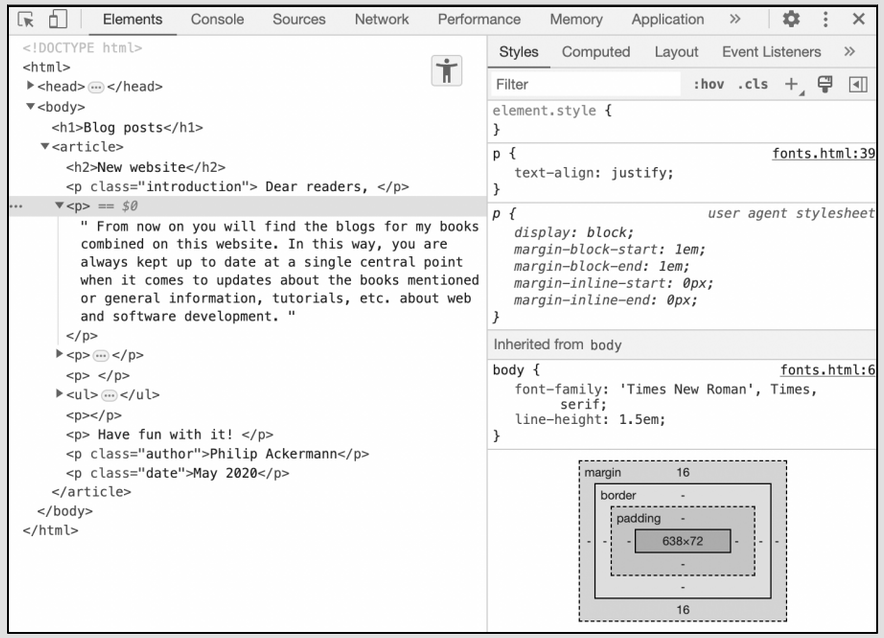

Figure 3.6 CSS Rules That Apply to an HTML Element

The **Computed** tab, on the other hand, provides information about which properties have been assigned to an element, as shown in Figure 3.7. This information is interesting, for example, if multiple CSS rules apply to an element.

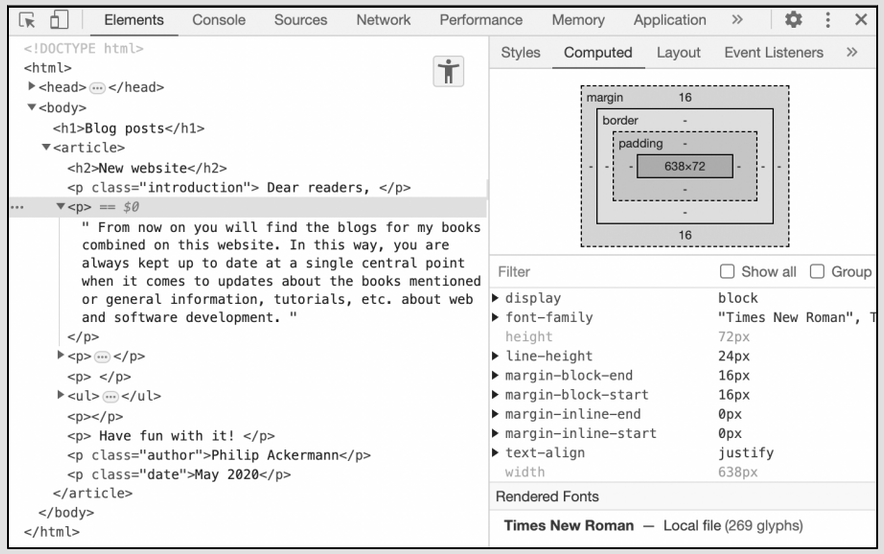

Figure 3.7 CSS Properties Applied to an HTML Element

3.3 Lists and Tables

Components like lists and tables can also be customized using CSS. In this section, we'll look at what CSS properties are available for these elements.

3.3.1 Designing Lists

In the case of lists, you can primarily customize the bullets of individual list entries. Use the `list-style-type` property to specify their appearance. Depending on whether the list is an unordered list or an ordered list, different values are possible for this property.

Formatting Unordered Lists

Unordered lists are lists where the individual list entries are not sorted. By default, each list entry is preceded by a black dot as a bullet. For this type of lists, you can choose between the following values for the `list-style-type` property:

- `none`: No bullet is used.
- `disc`: The bullet is a black circle.
- `circle`: The bullet is a white circle with a black border.
- `square`: The bullet character is a black square.

Listing 3.18 and Figure 3.8 show a simple example of customizing bullets.

```
<!DOCTYPE html>
<html>
<head>
  <title>Formatting unordered lists</title>.
  <style type="text/css">
  ul.web-technologies li {
    list-style-type: circle;
  }
  <style>
  <meta charset="UTF-8">
</head>
<body>
<article>
  <p>
    <ul class="web-technologies">
      <li>HTML</li>
      <li>CSS</li>
      <li>JavaScript</li>
      <li>Node.js</li>
      <li>Docker</li>
    </ul>
```

```
  </p>
</article>
</body>
</html>
```

Listing 3.18 Formatting Unordered Lists with CSS

- HTML
- CSS
- JavaScript
- Node.js
- Docker

Figure 3.8 Format of an Unordered List

Designing Ordered Lists

Ordered lists are lists where the individual list entries are sorted in some way. For this type of list, to manipulate the appearance of the bullets and thus the appearance of the sorting, you can use the following values for the `list-style-type` property (among others):

- `decimal`: Enumeration in decimal numbers (1, 2, 3, ...)
- `decimal-leading-zero`: Enumeration in decimal numbers preceded by 0 for 1-digit numbers (01, 02, 03, ... 09, 10, 11, ...)
- `lower-alpha`: Enumeration in lowercase letters of the alphabet (a, b, c, ...)
- `upper-alpha`: Enumeration in uppercase letters of the alphabet (A, B, C, ...)
- `lower-roman`: Enumeration in Roman numerals, represented by lowercase letters (i, ii, iii, ...)
- `upper-roman`: Enumeration in Roman numerals, represented by uppercase letters (I, II, III, ...)

In Listing 3.19, for example, bullets are presented as Roman numerals. The result is shown in Figure 3.9.

```
<!DOCTYPE html>
<html>
<head>
  <title>Format ordered lists</title>.
  <style type="text/css">
  ol.web-technologies li {
    list-style-type: lower-roman;
  }
  <style>
  <meta charset="UTF-8">
</head>
```



```
<body>
<article>
  <p>
    <ol class="web-technologies">
      <li>HTML</li>
      <li>CSS</li>
      <li>JavaScript</li>
      <li>Node.js</li>
      <li>Docker</li>
    </ol>
  </p>
</article>
</body>
</html>
```

Listing 3.19 Formatting Ordered Lists with CSS

i. HTML
ii. CSS
iii. JavaScript
iv. Node.js
v. Docker

Figure 3.9 Format of an Ordered List

Using Images as Bullets

In addition to configuring bullets via the `list-style-type` property, for unordered lists, you also have the option of defining a graphic to be used as the bullet. You can pass the path or URL for the desired image to the corresponding `list-style-image` property, as shown in Listing 3.20. Then, the image is used as a bullet for list entries, as shown in Figure 3.10.

```
<!DOCTYPE html>
<html>
<head>
  <title>Use images as bullets</title>.
  <style type="text/css">
  ul.books li {
    list-style-image: url("images/star.png");
  }
  <style>
  <meta charset="UTF-8">
</head>
<body>
<article>
  <p>
```

```
    <ul class="books">
      <li>Cal Newport: "Deep Work"</li>
      <li>James Clear: "Atomic Habits"</li>
      <li>Jake Knapp, John Zeratsky: "Make Time"</li>
      <li>Greg McKeown: "Essentialism"</li>
      <li>Nir Eyal: "Indistractable"</li>
    </ul>
  </p>
</article>
</body>
</html>
```

Listing 3.20 Using Images as Bullets

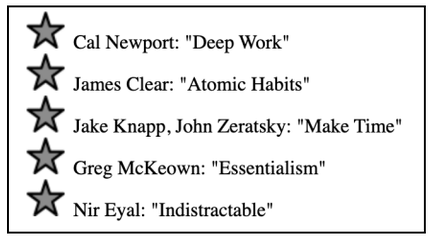

Figure 3.10 Using CSS to Customize Bullet Lists

Setting the Position of Bullet Points

You can use the `list-style-position` property to customize the position of the bullet points. The `outside` value ensures that the bullet points are to the left of the text block, which is also the default if the position is not controlled via CSS. The `inside` value, on the other hand, ensures that the bullet points are positioned indented within the text block (this value works for both unordered and ordered lists).

```
<!DOCTYPE html>
<html>
<head>
  <title>Indent bullets</title>
  <meta charset="utf-8">
  <style type="text/css">
  ul.web-technologies {
    list-style-position: inside;
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    width: 300px;
  }
  <style>
  <meta charset="UTF-8">
</head>
```



```
<body>
<article>
  <p>
    <ul class="web-technologies">
      <li>HTML: Hypertext Markup Language</li>
      <li>CSS: Cascading Style Sheets</li>
      <li>JavaScript: THE language of the web</li>
      <li>Node.js: Runtime environment for JavaScript</li>
      <li>Docker: Container virtualization software</li>
    </ul>
  </p>
</article>
</body>
</html>
```

Listing 3.21 Setting the Position of Bullets

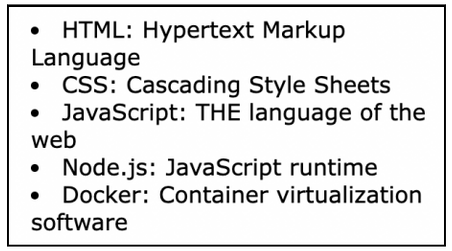

Figure 3.11 Displaying Indented Bullets with the Inside Value

3.3.2 Designing Tables

Tables always look rather plain at first without CSS, as shown in Figure 3.12.

Recommended books on CSS

Author	Title	Year of publication
Keith J. Grant	CSS in Depth	2018
Eric A. Meyer	CSS Pocket Reference: Visual Presentation for the Web	2018
Eric Meyer & Estelle Weyl	CSS: The Definitive Guide: Visual Presentation for the Web	2017
Lea Verou	CSS Secrets: Better Solutions to Everyday Web Design Problems	2014
Peter Gasston	The Book of CSS3: A Developer's Guide to the Future of Web Design	2014

Figure 3.12 Tables That Aren't Formatted with CSS Don't Really Look Nice by Default

The good news is that the appearance of tables can be easily customized using CSS. In this section, I want to show you briefly which CSS properties are used to make the table shown in Figure 3.12 more descriptive and to get to the result shown in Figure 3.13.

Recommended books on CSS

Author	Title	Year of publication
Keith J. Grant	*CSS in Depth*	2018
Eric A. Meyer	*CSS Pocket Reference: Visual Presentation for the Web*	2018
Eric Meyer & Estelle Weyl	*CSS: The Definitive Guide: Visual Presentation for the Web*	2017
Lea Verou	*CSS Secrets: Better Solutions to Everyday Web Design Problems*	2014
Peter Gasston	*The Book of CSS3: A Developer's Guide to the Future of Web Design*	2014

Figure 3.13 Tables Formatted with CSS Are More Appealing

> **Note**
>
> Most of the CSS properties presented in this section can be applied to other elements in addition to tables.

You already know some CSS properties commonly used for table design, for example, for adjusting the font (`font-family`), the font style (`font-style` and `font-weight`), the text alignment (`text-align`), the text color (`color`), and the background color (`background-color`). Thus, I won't discuss these properties further.

In Listing 3.22, I've highlighted new properties accordingly. Also highlighted are the pseudo-classes (or their corresponding selectors) that have not yet been mentioned. Let's go through these properties and pseudo-classes in order of occurrence:

- The `border` property is used to *design borders*, in this case, to design the border of the entire table. This property allows you to specify the width of the border (`thin` but pixel specifications are also allowed), the style (`solid`), and the color (`#000000`).

> **Shorthand Properties**
>
> The `border` property is a special kind of CSS property, called a *shorthand property*, because this property combines several other CSS properties: `border-width` (for setting the width of the border), `border-style` (for setting the style), and `border-color` (for setting the color). These properties in turn are also shorthand properties: For example, `border-width` is a shorthand property for the `border-top-width` (width of the top border), `border-right-width` (width of the right border), `border-bottom-width` (width of the bottom border), and `border-left-width` (width of the left border) properties. The shorthand property for designing borders is always useful when you want the border to look the same for all pages. This approach will save you some typing (the corresponding written out variants are also included in the listing for demonstration purposes).

- Usually, each cell of a table has its own border. You can use the `border-collapse` property to "collapse" these borders (`collapse` value), that is, that the individual cells "share" a border.
- You can use the `padding` property to define how much space should exist be between the content of a cell and its border. This property allows you to make the table more "airy" and less squat than the default.
- With the pseudo-class `:first-child`, you can select the first child element of a given element type. For example, the `td:first-child` selector selects all first `<td>` child elements, in other words, all cells in the first column. The selector in our example will print the text in the first column in bold.
- The `:nth-child()` pseudo-class, on the other hand, can select child elements that are located at a specific position within the parent element (the position is passed as a parameter). In our example, we are selecting the cells of the second column to display these cells in italics and selecting the cells of the third column to align the text of these cells to the right. In addition, `:nth-child(odd)` and `:nth-child(even)` can be used to select child elements that are at an odd position (`odd`) or at an even position (`even`). In our example, we colored "odd" table rows with a different background color from the "even" table rows.

Thus, with relatively little effort, tables can be made a lot more appealing than their default look.

```
<!DOCTYPE html>
<html>
<head>
  <title>Formatting Tables</title>
  <style type="text/css">

  body {
    font-family: Verdana, sans-serif;
  }

  table {
    /* Thin, solid, black border */
    border: thin solid #000000;

    /* Alternative, but more typing work: */
    /*
    border-width: thin;
    border-style: solid;
    border-color: #000000;
    */
```

```
  /* Alternative, but even more typing work: */
  /*
  border-top-width: thin;
  border-right-width: thin;
  border-bottom-width: thin;
  border-left-width: thin;
  border-top-style: solid;
  border-right-style: solid;
  border-bottom-style: solid;
  border-left-style: solid;
  border-top-color: #000000;
  border-right-color: #000000;
  border-bottom-color: #000000;
  border-left-color: #000000;
  */

  /* No double borders
     for adjacent cells */
  border-collapse: collapse;
}

/* Table headers */
th {
  background-color: #000000;
  color: #FFFFFF;
  text-align: left;
}

/* Table headers and cells */
th, td {
  padding: 11px;
}

/* Odd rows */
tr:nth-child(odd) {
  background-color: #CCCCCC;
}

/* Even rows */
tr:nth-child(even) {
  background-color: #FFFFFF;
}

/* First column of table */
```



```
  td:first-child {
    font-weight: bold;
  }

  /* Second column of table */
  td:nth-child(2) {
    font-style: italic;
  }

  /* Third column of table */
  td:nth-child(3) {
    text-align: right;
  }

  <style>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Recommended books on CSS</h1>
  <table>
    <thead>
      <tr>
        <th>Author</th>
        <th>Title</th>
        <th>Year of publication</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>
          Keith J. Grant
        </td>
        <td>
          CSS in Depth
        </td>
        <td>2018</td>
      </tr>
      <tr>
        <td>
         Eric A. Meyer
        </td>
        <td>
          CSS Pocket Reference: Visual Presentation for the Web
        </td>
```

```
      <td>2018</td>
    </tr>
    <tr>
      <td>
        Eric Meyer & Estelle Weyl
      </td>
      <td>
        CSS: The Definitive Guide: Visual Presentation for the Web
      </td>
      <td>2017</td>
    </tr>
    <tr>
      <td>
        Lea Verou
      </td>
      <td>
        CSS Secrets: Better Solutions to Everyday Web Design Problems
      </td>
      <td>2014</td>
    </tr>
    <tr>
      <td>
        Peter Gasston
      </td>
      <td>
        The Book of CSS3: A Developer's Guide to the Future of Web Design
      </td>
      <td>2014</td>
    </tr>
  </tbody>
 </table>
</body>
</html>
```

Listing 3.22 Designing Tables with CSS

3.4 Understanding the Different Layout Systems

Using CSS, you can precisely position HTML elements on a web page and thus influence the layout of the web page. In the early days of the web, tables were often more or less misused for positioning elements (using them as *layout tables*), although tables in HTML should only be used for displaying table data. Gradually, other possibilities were created (called *layout systems*), which I'll describe in this section in their order of appearance.

3.4.1 Basic Principles of Positioning with CSS

Before taking a closer look at layout systems, let's cover some basic principles of positioning elements.

Block-Level Elements and Inline-Level Elements

Basically, HTML distinguishes between *block-level elements* and *inline-level elements*. Block-level elements are always displayed by the browser on a new line, whereas inline-level elements are displayed where they appear in the text flow, as shown in Figure 3.14. In other words, block-level elements are arranged *vertically*, inline-level elements, *horizontally*.

However, with the CSS property `display`, you can customize the behavior of elements with regard to text flow: The `block` value ensures that the element is considered a block-level element; the `inline` value accordingly ensures that the element is considered an inline-level element.

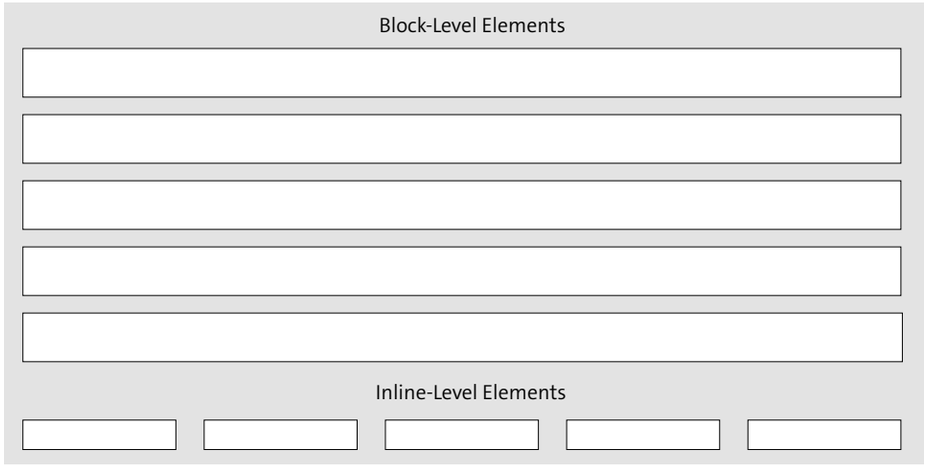

Figure 3.14 Block-Level Elements versus Inline-Level Elements

Types of Positioning

Via CSS, you have basically different ways to position elements—no matter if block-level or inline-level, as shown in Figure 3.15.

This positioning can be modified via the `position` property:

- Static positioning (`static` value): In this case, the elements are positioned as they appear in the HTML code (in the "normal flow").
- Relative positioning (`relative` value): In this case, elements are positioned relatively upwards, to the right, downwards, or to the left based on their position in the normal flow.

- Absolute positioning (`absolute` value): In this case, elements are taken out of the normal flow and positioned in relation to the parent element.
- Fixed positioning (`fixed` value): In this case, elements are positioned relative to the browser window (also called the *viewport*).
- Sticky positioning (`sticky` value): In this case, elements behave similarly to fixed positioning in terms of positioning but scroll only to a specified point and then remain fixed in the viewport.
- Inherited positioning (`inherit` value): In this case, the positioning behavior is inherited from the parent element.

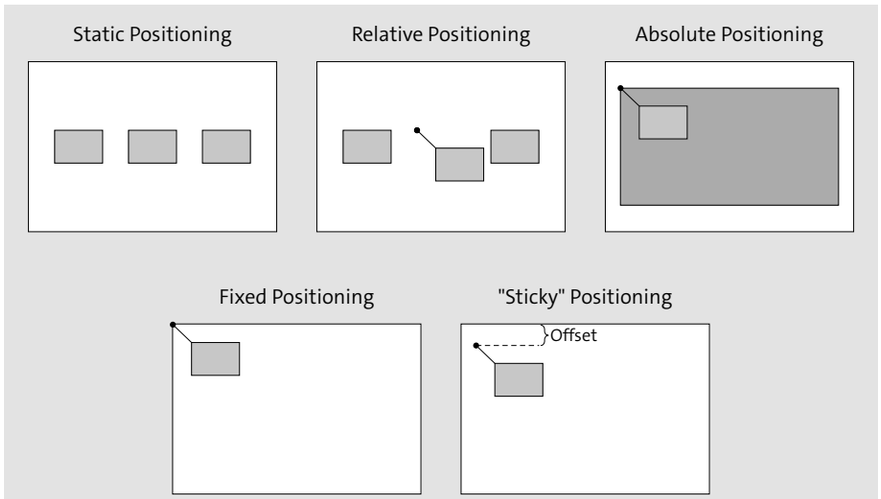

Figure 3.15 Comparing Positions in CSS

In addition, elements can be positioned using the layout systems mentioned earlier.

3.4.2 Float Layout

You can use the *float layout* to influence the behavior of elements in relation to the text flow. The corresponding CSS property `float` can be used to set elements to the left or right border of the surrounding HTML block, as shown in Figure 3.16.

For a long time, the float layout was the weapon of choice for arranging elements. Now, however, you have two alternatives—the *flexbox layout* and the *grid layout*—which are more suitable and which I want to describe in the next two sections. Nevertheless, I'd first like to explain the float layout using forms as an example. We'll then implement the same form with the other two layout systems, and in this way, you can directly evaluate the different layouts.

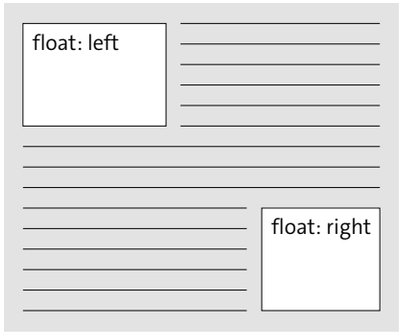

Figure 3.16 The Float Layout Principle

Example: Float Layout for Forms

Forms look even worse than tables by default, as shown in Figure 3.17. However, the individual form elements behave normally: They "flow" with the text flow from left to right and from top to bottom because form elements, such as `<label>`, `<input>`, and `<button>`, are all inline-level elements.

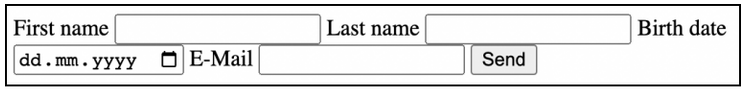

Figure 3.17 The Readability of Forms Is Not Particularly Good by Default

Forms can be customized using the float layout, as shown in Listing 3.23. Using the `float` property, the `<label>`, `<input>`, and `<button>` elements are aligned to the left (`left`) and right (`right`) of the surrounding `<form>` element. The `overflow` property in the `<form>` element also ensures that this element is extended in height accordingly to include all "flowing" elements (if you were to omit this property, these elements would protrude above the `<form>` element, which would then be too small).

In addition, some other CSS properties can affect the overall appearance of a form, such as the background color, rounded corners for the border of the form, text alignment, and more, but do not otherwise contribute to the positioning of the elements.

However, the `width` and `max-width` properties are still important, as these properties enable you to specify a preferred width as well as a maximum width for elements. In our example, we defined a maximum width for the form as well as defined the widths of the labels and text fields in each case.

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

```
<title>Design Forms</title>
<style type="text/css">

  body {
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    font-size: 0.9em;
  }

  * {
    box-sizing: border-box;
  }

  form {
    padding: 1em;
    background: #f9f9f9;
    border: 1px solid lightgrey;
    margin: 2rem auto auto auto;
    max-width: 600px;
    padding: 1em;
    border-radius: 5px;
    overflow: hidden;
  }

  form input {
    margin-bottom: 1rem;
    background: white;
    border: 1px solid darkgray;
  }

  form button {
    background: lightgrey;
    padding: 0.8em;
    border: 0;
  }

  form button:hover {
    background: deepskyblue;
  }

  label {
    text-align: right;
    display: block;
```



```
    padding: 0.5em 1.5em 0.5em 0;
  }

  input {
    width: 100%;
    padding: 0.7em;
    margin-bottom: 0.5rem;
  }

  input:focus {
    outline: 3px solid deepskyblue;
  }

  label {
    float: left;
    width: 200px;
  }

  input {
    float: left;
    width: calc(100% - 200px);
  }

    button {
      float: right;
      width: calc(100% - 200px);
    }

  <style>
  <meta charset="UTF-8">
</head>
<body>
  <form>
    <label for="firstName">First name</label>
    <input id="firstName" type="text">

    <label for="lastName">Last name</label>
    <input id="lastName" type="text">

    <label for="birthday">Birth date</label>
    <input id="birthday" type="date">
```

```
    <label for="mail">Email:</label>
    <input id="email" type="email">

    <button>Send</button>
</form>
</body>
</html>
```

Listing 3.23 Designing Forms Using the Float Layout

Thanks to these adjustments, the form already looks much better.

Figure 3.18 A Form Designed with CSS

3.4.3 Flexbox Layout

The *flexbox layout*, introduced with CSS3, is much more flexible than the float layout or the standard layout of block-level elements or inline-level elements. The main concept behind the flexbox layout is to allow an element to dynamically adjust the width and height of its child elements to best fill the available space. A *flex container* accordingly expands the width and height of the child elements (the *flex items*) to fill the available free space or shrinks them to prevent "overflow." Unlike the regular layouts of (vertically arranged) block-level elements and (horizontally arranged) inline-level elements, the flexbox layout is *direction independent*. In other words, this layout can be used for both vertical arrangements and horizontal arrangements, as shown in Figure 3.19.

You can use various CSS properties to influence the display of the individual flex items, including alignment (shown in Figure 3.20), expansion (shown in Figure 3.21), and arrangement (shown in Figure 3.22).

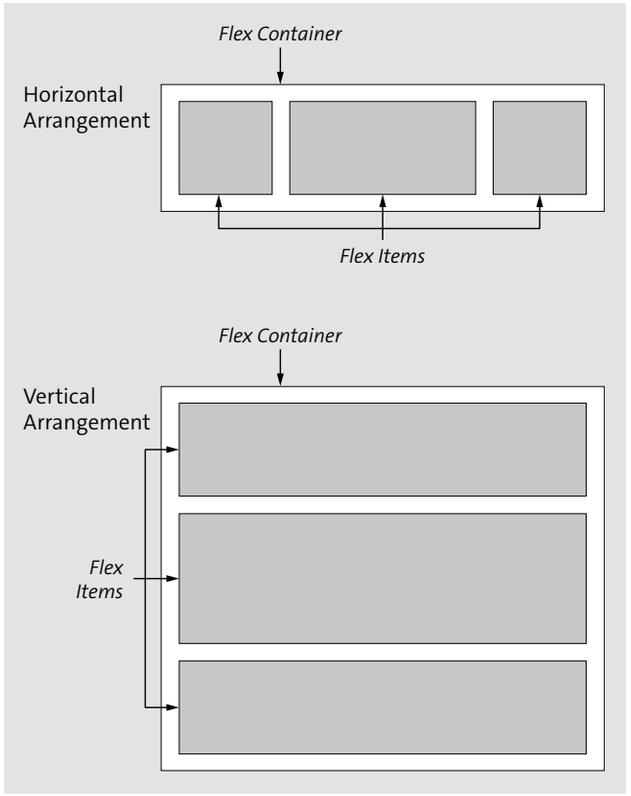

Figure 3.19 Flexbox Layout Allowing Horizontal and Vertical Arrangements of Elements

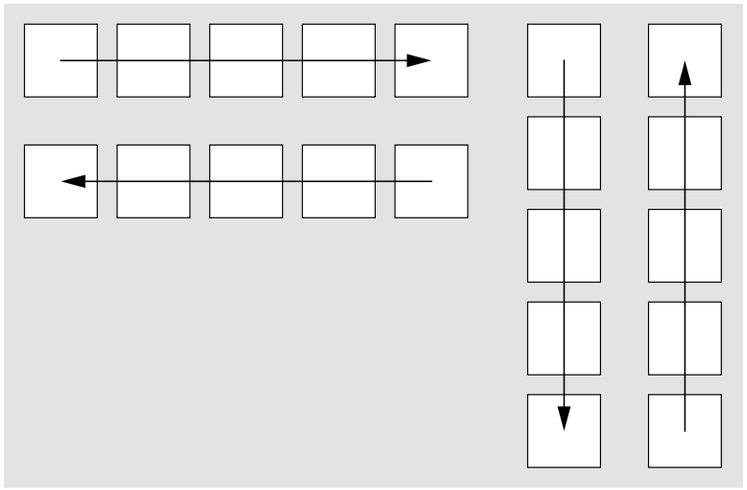

Figure 3.20 With the Flexbox Layout, Alignment Can Be Adjusted

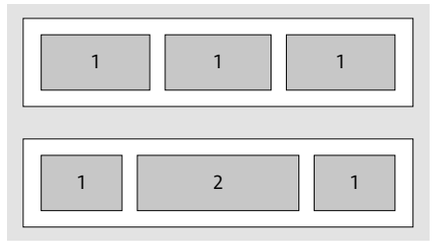

Figure 3.21 With the Flexbox Layout, Individual Items Can Be Extended

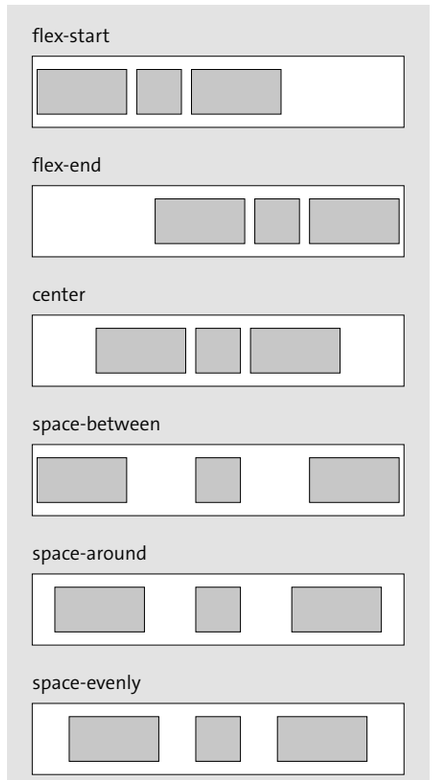

Figure 3.22 With the Flexbox Layout, Arrangements within the Flex Container Can Be Adjusted

Table 3.2 provides an overview of the CSS properties relevant to the flexbox layout.

Property	Refers to...	Description
`display`	Flex container	Defines an element as a flex container with value `flex`.

Table 3.2 CSS Properties Related to the Flexbox Layout

Property	Refers to...	Description
Sequence and Arrangement		
`flex-direction`	Flex container	Specifies the direction in which flex items are arranged in a flex container.
`flex-wrap`	Flex container	Controls whether the flex container is single-line or multi-line.
`flex-flow`	Flex container	Shorthand property for `flex-direction` and `flex-wrap` properties.
`order`	Flex item	Determines the order of individual flex items. A numeric value representing the position of the respective flex item can be specified as the value.
Alignment		
`justify-content`	Flex container	Defines how the available space should be distributed among flex items.
`align-items`	Flex container	Sets the default orientation of all elements for the other axis of the flex container.
`align-self`	Flex item	Like `align-items` but refers to individual flex items.
`align-content`	Multiline flex container	Defines the arrangement of individual flex items in multiline flex containers.
Extension		
`flex-basis`	Flex item	Controls via a numeric value how large an element can become along the major axis before it grows or shrinks.
`flex-grow`	Flex item	Determines via a numeric value how much an element can grow in relation to its sibling elements.
`flex-shrink`	Flex item	Determines via a numeric value by how much an element shrinks in relation to its sibling elements.
`flex`	Flex item	Shorthand property for the previous three properties.

Table 3.2 CSS Properties Related to the Flexbox Layout (Cont.)

Now, you have the essential basics in a nutshell. Let's see how a form can be implemented using the flexbox layout next.

Example: Flexbox Layout for Forms

First, a few adjustments to the HTML code are necessary: The individual labels and text fields are each placed in pairs in `<div>` elements. These elements are then assigned the `display` property and its value is set to `flex`, which results in each of these elements being interpreted as its own flexbox container. Labels and text fields are then arranged within each container, with the `flex` property defining their extent: Text fields (value `2`) have twice the width of labels (value `1`). The `justify-content` property with value `flex-end` also ensures that the elements within each container are arranged at the end (in this case on the right). The result of all this code is shown in Figure 3.23.

```
<!DOCTYPE html>
<html>
<head>
  <title>Design Forms</title>
  <style type="text/css">
    body {
      font-family: Verdana, Geneva, Tahoma, sans-serif;
      font-size: 0.9em;
    }

    form {
      padding: 1em;
      background: #f9f9f9;
      border: 1px solid lightgrey;
      margin: 2rem auto auto auto;
      max-width: 600px;
      border-radius: 5px;
    }

    form input {
      margin-bottom: 1rem;
      background: white;
      border: 1px solid darkgray;
    }

    form button {
      background: lightgrey;
      padding: 0.8em;
      border: 0;
    }

    form button:hover {
      background: deepskyblue;
    }
```



```
    label {
      text-align: right;
      display: block;
      padding: 0.5em 1.5em 0.5em 0;
    }

    input {
      width: 100%;
      padding: 0.7em;
      margin-bottom: 0.5rem;
    }

    input:focus {
      outline: 3px solid deepskyblue;
    }

    form {
      overflow: hidden;
    }

    .form-row {
      display: flex;
      justify-content: flex-end;
    }

    .form-row > label {
      flex: 1;
    }

    .form-row > input {
      flex: 2;
    }

  </style>
  <meta charset="UTF-8">
</head>
<body>
  <form>
    <div class="form-row">
      <label for="firstName">First name</label>
      <input id="firstName" type="text">
    </div>
```

```
    <div class="form-row">
      <label for="lastName">Last name</label>
      <input id="lastName" type="text">
    </div>

    <div class="form-row">
      <label for="birthday">Birth date</label>
      <input id="birthday" type="date">
    </div>

    <div class="form-row">
      <label for="mail">Email:</label>
      <input id="email" type="email">
    </div>

    <div class="form-row">
      <button>Send</button>
    </div>
  </form>
</body>
</html>
```

Listing 3.24 Designing Forms Using the Flexbox Layout

Figure 3.23 A Form Designed Using the Flexbox Layout

3.4.4 Grid Layout

The flexbox layout enables the flexible arrangement of elements. For complex layouts, however, the newer *grid layout* is more suitable. In contrast to the *one-dimensional* flexbox layout, this *two-dimensional* layout involves the positioning of elements (the *grid items*) in a (two-dimensional) grid inside a *grid container*, as shown in Figure 3.24. Individual elements arranged using the grid layout can therefore be arranged in two dimensions rather than just one, as is the case with the flexbox layout, making it much easier to implement complex layouts.

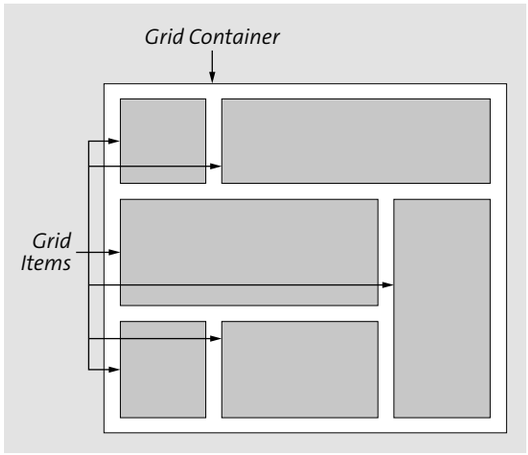

Figure 3.24 The Grid Layout Principle

As with the flexbox layout, a whole set of CSS properties are relevant for the grid layout. Table 3.3 provides an overview of these properties.

Property	Refers to...	Description
`display`	Grid container	Defines an element as a grid container with value `grid`.
Grid Definition		
`grid-template-columns`	Grid container	Defines the number and size of the columns of the grid.
`grid-template-rows`	Grid container	Defines the number and size of the rows of the grid.
`grid-auto-columns`	Grid container	Defines the number and size of automatically created columns of the grid.
`grid-auto-rows`	Grid container	Defines the number and size of automatically created rows of the grid.
`grid-auto-flow`	Grid container	Defines how automatically placed elements “flow” within the grid.

Table 3.3 CSS Properties Relevant to the Grid Layout

Property	Refers to...	Description
grid-template-areas	Grid container	Defines the structure and position of grid elements defined via grid-area.
grid-template	Grid container	Shorthand property for grid-template-areas, grid-template-columns, and grid-template-rows.
grid	Grid container	Shorthand property for grid-auto-columns, grid-auto-flow, grid-auto-rows, and grid-template.
Placement of Grid Items		
grid-row-start	Grid item	Defines the vertical start of a grid area.
grid-column-start	Grid item	Defines the horizontal start of a grid area.
grid-row-end	Grid item	Defines the vertical end of a grid area.
grid-column-end	Grid item	Defines the horizontal end of a grid area.
grid-row	Grid item	Shorthand property for grid-row-start and grid-row-end.
grid-column	Grid item	Shorthand property for grid-column-start and grid-column-end.
grid-area	Grid item	Shorthand property for grid-row and grid-column.
Alignment		
justify-self	Grid item	Defines the alignment of a grid item within the grid container.
justify-items	Grid container	Defines the alignment of all grid items within the grid container.
align-self	Grid item	Defines for a grid item how a single grid item is positioned along the cross axis.
align-items	Grid container	Defines for a grid container how the individual grid items are positioned along the cross axis.
justify-content	Grid container	Defines the direction of the main axis along which grid items are oriented.
align-content	Grid container	Defines how individual grid items are positioned along the cross axis (the cross axis is defined as whichever axis is not the main axis).

Table 3.3 CSS Properties Relevant to the Grid Layout (Cont.)

Property	Refers to...	Description
Blank Space		
row-gap	Grid container	Defines the distances between the individual rows of a grid.
column-gap	Grid container	Defines the distances between the individual columns of a grid.
gap	Grid container	Shorthand property for row-gap and column-gap.

Table 3.3 CSS Properties Relevant to the Grid Layout (Cont.)

Example: Grid Layout for Forms

Even though our form example does not use the two-dimensional characteristic of the grid layout to its fullest extent, I hope that my example shows you how easy using this layout system is. First, as shown in Listing 3.25, note that the `<div>` elements necessary for the flexbox layout have been omitted. The `<form>` element is defined as a grid container via the `display` property. Via `grid-template-columns`, the number and width of the columns of the grid are defined: The number results from the number of values (separated by spaces), whereas the width is given as a fraction (`fr` for "fraction"). So, the value `1fr 2fr` defines two columns, with the second column twice as wide as the first. You can also use the `grid-gap` property to define the spacing between the columns.

Then, using the `grid-column` property for the labels, text fields, and button, you can define exactly from which column to which column the elements should be arranged: the labels from column 1 to column 2 (`1 / 2`) and the text fields and the button from column 2 to column 3 (`2 / 3`). The result of all this code is shown in Figure 3.25.

```
<!DOCTYPE html>
<html>
<head>
  <title>Design Forms</title>
  <style type="text/css">

    body {
      font-family: Verdana, Geneva, Tahoma, sans-serif;
      font-size: 0.9em;
    }

    form {
      display: grid;
      grid-template-columns: 1fr 2fr;
      grid-gap: 16px;
```

```
  background: #f9f9f9;
  border: 1px solid lightgrey;
  margin: 2rem auto 0 auto;
  max-width: 600px;
  padding: 1em;
  border-radius: 5px;
}

form input {
  background: white;
  border: 1px solid darkgray;
}

form button {
  background: lightgrey;
  padding: 0.8em;
  width: 100%;
  border: 0;
}

form button:hover {
  background: deepskyblue;
}

label {
  padding: 0.5em 0.5em 0.5em 0;
  text-align: right;
  grid-column: 1 / 2;
}

input {
  padding: 0.7em;
}

input:focus {
  outline: 3px solid deepskyblue;
}

input,
button {
  grid-column: 2 / 3;
}
```



```
  <style>
  <meta charset="UTF-8">
</head>
<body>
  <form>
    <label for="firstName">First name</label>
    <input id="firstName" type="text">

    <label for="lastName">Last name</label>
    <input id="lastName" type="text">

    <label for="birthday">Birth date</label>
    <input id="birthday" type="date">

    <label for="mail">Email:</label>
    <input id="email" type="email">

    <button>Send</button>
</form>
</body>
</html>
```

Listing 3.25 Designing Forms Using the Grid Layout

Figure 3.25 A Form Designed Using the Grid Layout

3.5 Summary and Outlook

The basic principles of CSS are not particularly complex, as we've shown in this chapter, although implementing a particular layout in real life can be quite time-consuming in some circumstances. So, you won't get practical experience with CSS overnight. The

only thing that helps in this area is to try it out and look at many, many examples. I hope I provided some useful examples in this chapter and that you feel motivated to deal more extensively with this important language.

3.5.1 Key Points

Let's summarize a few key points you should take away from this chapter:

- *CSS rules* allow you to define how the content of certain HTML elements should be displayed.
- CSS rules basically consist of two parts: You can use the *selector* to define which HTML elements a CSS rule should be applied to. You can use the *declaration* to specify exactly how these HTML elements should be displayed.
- Individual declarations in turn consist of a *property* and a value.
- You can include CSS in an HTML document in several ways, namely, the following:
  - External stylesheets: In this case, you save CSS instructions as a separate file and include this file in the HTML document.
  - Internal stylesheets: In this case, you define the CSS instructions in the header of the HTML document.
  - Inline styles: In this case, you specify the CSS instructions directly in an HTML element.
- You can use CSS to design all the components of a web page. For texts, for example, the font, font style, text color, and alignment can be adjusted. We also showed you how to make lists, tables, and forms look appealing with CSS.
- CSS provides several layout systems for arranging elements:
  - In the *float layout*, elements "flow" in the text flow, and you can interrupt this flow and arrange elements in new lines this way.
  - With the *flexbox layout*, you can arrange elements in rows or columns and, among other things, specify the space that the elements take up in the process. The flexbox layout is a one-dimensional layout.
  - In the *grid layout*, elements can be arranged in grids of any complexity. This two-dimensional layout is the most flexible of the layout systems we've mentioned.

3.5.2 Recommended Reading

To study CSS in more detail, I recommend the following books:

- Keith J. Grant: *CSS in Depth* (2018)
- Eric A. Meyer: *CSS Pocket Reference: Visual Presentation for the Web* (2018)
- Eric Meyer & Estelle Weyl: *CSS: The Definitive Guide: Visual Presentation for the Web* (2017)

- Lea Verou: *CSS Secrets: Better Solutions to Everyday Web Design Problems* (2014)
- Peter Gasston: *The Book of CSS3: A Developer's Guide to the Future of Web Design* (2014)

Attentive readers will notice that this exact list was featured in our code example for designing tables in Section 3.3.2.

3.5.3 Outlook

Of course, this chapter is only an introduction to the CSS language. The language has a lot more to offer, including the following:

- Other features: CSS provides numerous other properties that we have not discussed in this chapter, for example, for defining background images or determining the width and height of elements.
- Animations: CSS allows you to animate elements on a web page by defining transitions between different values of properties. For example, you can use this method to slowly fade in elements, make them larger or smaller, or change their background color.
- Responsive design: This term refers to the ability of a web page to adapt its content to different screen sizes (for desktops, smartphones, etc.). Crucial in this context is the role played by what are called *media queries*, which enable you to apply or disable CSS rules depending on certain factors such as screen size. Chapter 11 discusses this topic.
- CSS frameworks: Sometimes, a lot of effort is required to define CSS rules to make standard elements on a web page (such as form elements, tables, and lists) look appealing. *CSS frameworks* like Bootstrap (*https://getbootstrap.com*), Semantic UI (*https://semantic-ui.com*), or Materialize (*https://materializecss.com*) offer prebuilt stylesheets that style standard elements accordingly. All you need to do is include the appropriate CSS files for the framework and possibly prepare the HTML structure accordingly, and the web page presents itself in the corresponding design or layout.
- In Chapter 9, we'll also discuss what are called *CSS preprocessors*, which can save you a great deal of typing when working with CSS.

Now that you've learned the basics of the two languages HTML and CSS, I want to introduce you to the JavaScript language in the next chapter and show you how to make web pages more interactive.

Contents

Foreword 23
Preface 25

1 Understanding the Basics 29

1.1 Terminology 29
1.1.1 Client and Server 29
1.1.2 Relationship between URLs, Domains, and IP Addresses 30

1.2 Structure of Web Applications 32
1.2.1 Creating Web Pages Using HTML 32
1.2.2 Designing Web Pages with CSS 33
1.2.3 Making Web Pages Interactive with JavaScript 34
1.2.4 Making Web Pages Dynamic Using Server-Side Logic 35

1.3 Full Stack Development 36
1.3.1 What Are Software Stacks? 36
1.3.2 What Types of Stacks Exist? 37
1.3.3 What Is a Full Stack Developer? 38
1.3.4 Structure of This Book 40

1.4 Tools for Full Stack Developers 42
1.4.1 Editors 43
1.4.2 Development Environments 44
1.4.3 Browsers 46

1.5 Summary and Outlook 50
1.5.1 Key Points 50
1.5.2 Outlook 50

2 Structuring Web Pages with HTML 51

2.1 Introduction 51
2.1.1 Versions 52
2.1.2 Using Elements and Attributes 52
2.1.3 Saving Web Pages as HTML Documents 54

2.2 Using the Most Important Elements ... 56
2.2.1 Using Headings, Paragraphs, and Other Text Formatting ... 56
2.2.2 Creating Lists ... 57
2.2.3 Defining Links ... 59
2.2.4 Including Images ... 64
2.2.5 Structuring Data in Tables ... 66
2.2.6 Defining Forms ... 72
2.2.7 Further Information ... 76
2.3 Summary and Outlook ... 77
2.3.1 Key Points ... 77
2.3.2 Recommended Reading ... 78
2.3.3 Outlook ... 78

3 Designing Web Pages with CSS 79

3.1 Introduction ... 79
3.1.1 The Principle of CSS ... 80
3.1.2 Including CSS in HTML ... 80
3.1.3 Selectors ... 85
3.1.4 Cascading and Specificity ... 88
3.1.5 Inheritance ... 91
3.2 Applying Colors and Text Formatting ... 91
3.2.1 Defining the Text Color and Background Color ... 91
3.2.2 Designing Texts ... 93
3.3 Lists and Tables ... 103
3.3.1 Designing Lists ... 103
3.3.2 Designing Tables ... 107
3.4 Understanding the Different Layout Systems ... 112
3.4.1 Basic Principles of Positioning with CSS ... 113
3.4.2 Float Layout ... 114
3.4.3 Flexbox Layout ... 118
3.4.4 Grid Layout ... 124
3.5 Summary and Outlook ... 129
3.5.1 Key Points ... 130
3.5.2 Recommended Reading ... 130
3.5.3 Outlook ... 131

4 Making Web Pages Interactive with JavaScript 133

4.1 Introduction ... 133
4.1.1 Including JavaScript ... 134
4.1.2 Displaying Dialog Boxes ... 136
4.1.3 Using the Developer Console ... 137
4.1.4 Introduction to Programming ... 139
4.2 Variables, Constants, Data Types, and Operators ... 140
4.2.1 Defining Variables ... 140
4.2.2 Defining Constants ... 141
4.2.3 Using Data Types ... 141
4.2.4 Using Operators ... 143
4.3 Using Control Structures ... 144
4.3.1 Using Conditional Statements and Branching ... 144
4.3.2 Using Loops ... 146
4.4 Functions and Error Handling ... 147
4.4.1 Defining and Calling Functions ... 147
4.4.2 Passing and Analyzing Function Parameters ... 148
4.4.3 Defining Return Values ... 149
4.4.4 Responding to Errors ... 149
4.5 Objects and Arrays ... 151
4.5.1 Using Objects ... 151
4.5.2 Using Arrays ... 152
4.6 Summary and Outlook ... 154
4.6.1 Key Points ... 154
4.6.2 Recommended Reading ... 155
4.6.3 Outlook ... 155

5 Using Web Protocols 157

5.1 Hypertext Transfer Protocol ... 157
5.1.1 Requests and Responses ... 158
5.1.2 Structure of HTTP Requests ... 160
5.1.3 Structure of HTTP Responses ... 161
5.1.4 Header ... 162
5.1.5 Methods ... 164
5.1.6 Status Codes ... 166
5.1.7 MIME Types ... 167

5.1.8 Cookies ... 170
5.1.9 Executing HTTP from the Command Line ... 173

5.2 Bidirectional Communication ... 174
5.2.1 Polling and Long Polling ... 174
5.2.2 Server-Sent Events ... 175
5.2.3 WebSockets ... 176

5.3 Summary and Outlook ... 178
5.3.1 Key Points ... 178
5.3.2 Recommended Reading ... 178
5.3.3 Outlook ... 179

6 Using Web Formats 181

6.1 Data Formats ... 182
6.1.1 CSV ... 182
6.1.2 XML ... 182
6.1.3 JSON ... 187

6.2 Image Formats ... 193
6.2.1 Photographs in the JPG Format ... 193
6.2.2 Graphics and Animations in the GIF Format ... 193
6.2.3 Graphics in the PNG Format ... 194
6.2.4 Vector Graphics in the SVG Format ... 195
6.2.5 Everything Gets Better with the WebP Format ... 196
6.2.6 Comparing Image Formats ... 196
6.2.7 Programs for Image Processing ... 198

6.3 Video and Audio Formats ... 199
6.3.1 Video Formats ... 199
6.3.2 Audio Formats ... 202

6.4 Summary and Outlook ... 204
6.4.1 Key Points ... 204
6.4.2 Recommended Reading ... 204
6.4.3 Outlook ... 205

7 Using Web APIs 207

7.1 Changing Web Pages Dynamically Using the DOM API ... 208
7.1.1 The Document Object Model ... 208

7.1.2 The Different Types of Nodes ... 209
7.1.3 Selecting Elements ... 211
7.1.4 Modifying Elements ... 213
7.1.5 Creating, Adding, and Deleting Elements ... 214
7.1.6 Practical Example: Dynamic Creation of a Table ... 215

7.2 Loading Data Synchronously via Ajax and the Fetch API ... 218
7.2.1 Synchronous versus Asynchronous Communication ... 218
7.2.2 Loading Data via Ajax ... 220
7.2.3 Loading Data via the Fetch API ... 223

7.3 Other Web APIs ... 223
7.3.1 Overview of Web APIs ... 224
7.3.2 Browser Support for Web APIs ... 227

7.4 Summary and Outlook ... 227
7.4.1 Key Points ... 228
7.4.2 Recommended Reading ... 228
7.4.3 Outlook ... 228

8 Optimizing Websites for Accessibility 229

8.1 Introduction ... 229
8.1.1 Introduction to Accessibility ... 230
8.1.2 User Groups and Assistive Technologies ... 230
8.1.3 Web Content Accessibility Guidelines ... 232

8.2 Making Components of a Website Accessible ... 236
8.2.1 Structuring Web Pages Semantically ... 236
8.2.2 Using Headings Correctly ... 239
8.2.3 Making Forms Accessible ... 239
8.2.4 Making Tables Accessible ... 241
8.2.5 Making Images Accessible ... 246
8.2.6 Making Links Accessible ... 248
8.2.7 Accessible Rich Internet Applications ... 249
8.2.8 Miscellaneous ... 251

8.3 Testing Accessibility ... 254
8.3.1 Types of Tests ... 254
8.3.2 Tools for Testing ... 255

8.4 Summary and Outlook ... 258
8.4.1 Key Points ... 258
8.4.2 Recommended Reading ... 259
8.4.3 Outlook ... 259

9 Simplifying CSS with CSS Preprocessors 261

9.1 Introduction 261
9.1.1 How CSS Preprocessors Work 262
9.1.2 Features of CSS Preprocessors 262
9.1.3 Sass, Less, and Stylus 264

9.2 Using Sass 264
9.2.1 Installing Sass 264
9.2.2 Compiling Sass Files to CSS 265
9.2.3 Using Variables 266
9.2.4 Using Operators 270
9.2.5 Using Branches 271
9.2.6 Using Loops 272
9.2.7 Using Functions 276
9.2.8 Implementing Custom Functions 278
9.2.9 Nesting Rules 281
9.2.10 Using Inheritance and Mixins 282

9.3 Summary and Outlook 285
9.3.1 Key Points 285
9.3.2 Recommended Reading 286
9.3.3 Outlook 286

10 Implementing Single-Page Applications 287

10.1 Introduction 287

10.2 Setup 290

10.3 Components: The Building Blocks of a React Application 293
10.3.1 State: The Local State of a Component 295
10.3.2 The Lifecycle of a Component 296

10.4 Styling Components 298
10.4.1 Inline Styling 298
10.4.2 CSS Classes and External Stylesheets 299
10.4.3 Overview of Other Styling Options 301

10.5 Component Hierarchies 302

10.6 Forms 307

10.7 The Context API 310

10.8 Routing 314

10.9 Summary and Outlook 316
10.9.1 Key Points 317
10.9.2 Recommended Reading 317
10.9.3 Outlook 318

11 Implementing Mobile Applications 319

11.1 The Different Types of Mobile Applications 319
11.1.1 Native Applications 320
11.1.2 Mobile Web Applications 321
11.1.3 Hybrid Applications 323
11.1.4 Comparing the Different Approaches 324

11.2 Responsive Design 326
11.2.1 Introduction: What Is Responsive Design? 326
11.2.2 Viewports 328
11.2.3 Media Queries 330
11.2.4 Flexible Layouts 333

11.3 Cross-Platform Development with React Native 338
11.3.1 The Principle of React Native 338
11.3.2 Installation and Project Initialization 339
11.3.3 Starting the Application 340
11.3.4 The Basic Structure of a React Native Application 343
11.3.5 User Interface Components 344
11.3.6 Building and Publishing Applications 349

11.4 Summary and Outlook 349
11.4.1 Key Points 349
11.4.2 Recommended Reading 350
11.4.3 Outlook 350

12 Understanding and Using Web Architectures 351

12.1 Layered Architectures 352
12.1.1 Basic Structure of Layered Architectures 352
12.1.2 Client-Server Architecture (Two-Tier Architecture) 353
12.1.3 Multi-Tier Architecture 355

12.2 Monoliths and Distributed Architectures ... 358
12.2.1 Monolithic Architecture ... 358
12.2.2 Service-Oriented Architecture ... 359
12.2.3 Microservice Architecture ... 361
12.2.4 Component-Based Architecture ... 362
12.2.5 Microfrontends Architecture ... 363
12.2.6 Messaging Architecture ... 364
12.2.7 Web Service Architecture ... 366

12.3 MV* Architectures ... 367
12.3.1 Model-View-Controller ... 367
12.3.2 Model-View-Presenter ... 370
12.3.3 Model-View-Viewmodel ... 370

12.4 Summary and Outlook ... 371
12.4.1 Key Points ... 371
12.4.2 Recommended Reading ... 372
12.4.3 Outlook ... 372

13 Using Programming Languages on the Server Side 373

13.1 Types of Programming Languages ... 374
13.1.1 Programming Languages by Degree of Abstraction ... 374
13.1.2 Compiled and Interpreted Programming Languages ... 375

13.2 Programming Paradigms ... 378
13.2.1 Imperative and Declarative Programming ... 378
13.2.2 Object-Oriented Programming ... 379
13.2.3 Functional Programming ... 384

13.3 What Are the Programming Languages? ... 385
13.3.1 Rankings of Programming Languages ... 385
13.3.2 Which Programming Language Should You Learn? ... 388
13.3.3 But Seriously Now: Which Programming Language Should You Learn? ... 394

13.4 Summary and Outlook ... 395
13.4.1 Key Points ... 395
13.4.2 Recommended Reading ... 396
13.4.3 Outlook ... 397

14 Using JavaScript on the Server Side 399

14.1 JavaScript on Node.js ... 399
14.1.1 Node.js Architecture ... 400
14.1.2 A First Program ... 403
14.1.3 Package Management ... 405

14.2 Using the Integrated Modules ... 409
14.2.1 Reading Files ... 411
14.2.2 Writing Files ... 412
14.2.3 Deleting Files ... 413

14.3 Implementing a Web Server ... 413
14.3.1 Preparations ... 414
14.3.2 Providing Static Files ... 416
14.3.3 Using the Express.js Web Framework ... 420
14.3.4 Processing Form Data ... 421

14.4 Summary and Outlook ... 423
14.4.1 Key Points ... 424
14.4.2 Recommended Reading ... 424
14.4.3 Outlook ... 424

15 Using the PHP Language 425

15.1 Introduction to the PHP Language ... 425

15.2 Installing PHP and the Web Server Locally ... 425

15.3 Variables, Data Types, and Operators ... 427
15.3.1 Using Variables ... 428
15.3.2 Using Constants ... 432
15.3.3 Using Operators ... 432

15.4 Using Control Structures ... 435
15.4.1 Conditional Statements ... 435
15.4.2 Loops ... 437

15.5 Functions and Error Handling ... 439
15.5.1 Defining Functions ... 439
15.5.2 Function Parameters ... 439
15.5.3 Defining Return Values ... 441
15.5.4 Using Data Types ... 441

15.5.5 Anonymous Functions ... 442
15.5.6 Declaring Variable Functions ... 443
15.5.7 Arrow Functions ... 443
15.5.8 Responding to Errors ... 443

15.6 Using Classes and Objects ... 445
15.6.1 Writing Classes ... 445
15.6.2 Creating Objects ... 445
15.6.3 Class Constants ... 446
15.6.4 Visibility ... 446
15.6.5 Inheritance ... 447
15.6.6 Class Abstraction ... 448
15.6.7 More Features ... 449

15.7 Developing Dynamic Websites with PHP ... 450
15.7.1 Creating and Preparing a Form ... 450
15.7.2 Receiving Form Data ... 452
15.7.3 Verifying Form Data ... 452

15.8 Summary and Outlook ... 460
15.8.1 Key Points ... 460
15.8.2 Recommended Reading ... 461
15.8.3 Outlook ... 462

16 Implementing Web Services 463

16.1 Introduction ... 463

16.2 SOAP ... 465
16.2.1 The Workflow with SOAP ... 466
16.2.2 Description of Web Services with WSDL ... 467
16.2.3 Structure of SOAP Messages ... 469
16.2.4 Conclusion ... 470

16.3 REST ... 471
16.3.1 The Workflow with REST ... 471
16.3.2 The Principles of REST ... 472
16.3.3 Implementing a REST API ... 476
16.3.4 Calling a REST API ... 483

16.4 GraphQL ... 488
16.4.1 The Disadvantages of REST ... 488
16.4.2 The Workflow of GraphQL ... 491

16.5 Summary and Outlook ... 493
16.5.1 Key Points ... 493
16.5.2 Recommended Reading ... 494
16.5.3 Outlook ... 494

17 Storing Data in Databases 495

17.1 Relational Databases ... 496
17.1.1 The Functionality of Relational Databases ... 496
17.1.2 The SQL Language ... 498
17.1.3 Real-Life Example: Using Relational Databases in Node.js ... 506
17.1.4 Object-Relational Mappings ... 515

17.2 Non-Relational Databases ... 518
17.2.1 Relational versus Non-Relational Databases ... 518
17.2.2 The Functionality of Non-Relational Databases ... 519
17.2.3 Key-Value Databases ... 519
17.2.4 Document-Oriented Databases ... 520
17.2.5 Graph Databases ... 522
17.2.6 Column-Oriented Databases ... 523

17.3 Summary and Outlook ... 524
17.3.1 Key Points ... 524
17.3.2 Recommended Reading ... 525
17.3.3 Outlook ... 526

18 Testing Web Applications 527

18.1 Automated Tests ... 527
18.1.1 Introduction ... 528
18.1.2 Types of Tests ... 529
18.1.3 Test-Driven Development ... 531
18.1.4 Running Automated Tests in JavaScript ... 534

18.2 Test Coverage ... 537
18.2.1 Introduction ... 537
18.2.2 Determining Test Coverage in JavaScript ... 538

18.3 Test Doubles ... 539
18.3.1 The Problem with Dependencies ... 540
18.3.2 Replacing Dependencies with Test Doubles ... 540

18.3.3 Spies ... 541
18.3.4 Stubs ... 543
18.3.5 Mock Objects ... 543

18.4 Summary and Outlook ... 544
18.4.1 Key Points ... 544
18.4.2 Recommended Reading ... 545
18.4.3 Outlook ... 545

19 Deploying and Hosting Web Applications 547

19.1 Introduction ... 547
19.1.1 Building, Deploying, and Hosting ... 548
19.1.2 Types of Deployment ... 549
19.1.3 Types of Hosting ... 552
19.1.4 Requirements for Servers ... 555

19.2 Container Management ... 557
19.2.1 Docker ... 557
19.2.2 Real-Life Example: Packaging a Web Application using Docker ... 559
19.2.3 Number of Docker Images ... 565
19.2.4 Docker Compose ... 567

19.3 Summary and Outlook ... 569
19.3.1 Key Points ... 569
19.3.2 Recommended Reading ... 570
19.3.3 Outlook ... 570

20 Securing Web Applications 571

20.1 Vulnerabilities ... 572
20.1.1 Open Web Application Security Project ... 572
20.1.2 Injection ... 572
20.1.3 Broken Authentication ... 574
20.1.4 Sensitive Data Exposure ... 574
20.1.5 XML External Entities ... 575
20.1.6 Broken Access Control ... 575
20.1.7 Security Misconfiguration ... 576
20.1.8 Cross-Site Scripting ... 577
20.1.9 Insecure Deserialization ... 577

20.1.10 Using Components with Known Vulnerabilities ... 578
20.1.11 Insufficient Logging and Monitoring ... 579
20.1.12 Outlook ... 579

20.2 Encryption and Cryptography ... 579
20.2.1 Symmetric Cryptography ... 580
20.2.2 Asymmetric Cryptography ... 581
20.2.3 SSL, TLS, and HTTPS ... 582

20.3 Same-Origin Policies, Content Security Policies, and Cross-Origin Resource Sharing ... 584
20.3.1 Same Origin Policy ... 584
20.3.2 Cross-Origin Resource Sharing ... 585
20.3.3 Content Security Policy ... 587

20.4 Authentication ... 593
20.4.1 Basic Authentication ... 593
20.4.2 Session-Based Authentication ... 594
20.4.3 Token-Based Authentication ... 595

20.5 Summary and Outlook ... 597
20.5.1 Key Points ... 597
20.5.2 Recommended Reading ... 598
20.5.3 Outlook ... 598

21 Optimizing the Performance of Web Applications 599

21.1 Introduction ... 599
21.1.1 What Should Be Optimized and Why? ... 600
21.1.2 How Can Performance Be Measured? ... 601
21.1.3 Which Tools Are Available for Measuring Performance? ... 605

21.2 Options for Optimization ... 609
21.2.1 Optimizing Connection Times ... 609
21.2.2 Using a Server-Side Cache ... 611
21.2.3 Optimizing Images ... 612
21.2.4 Using a Client-Side Cache ... 615
21.2.5 Minifying the Code ... 618
21.2.6 Compressing Files ... 622
21.2.7 Lazy Loading: Loading Data Only When Needed ... 623
21.2.8 Preloading Data ... 623

21.3 Summary and Outlook 627
21.3.1 Key Points 628
21.3.2 Recommended Reading 629
21.3.3 Outlook 629

22 Organizing and Managing Web Projects 631

22.1 Types of Version Control Systems 632
22.1.1 Central Version Control Systems 632
22.1.2 Decentralized Version Control Systems 633
22.2 The Git Version Control System 635
22.2.1 How Git Stores Data 635
22.2.2 The Different Areas of Git 636
22.2.3 Installation 636
22.2.4 Creating a New Git Repository 638
22.2.5 Transferring Changes to the Staging Area 640
22.2.6 Committing Changes to the Local Repository 641
22.2.7 Committing Changes to the Remote Repository 643
22.2.8 Transferring Changes from the Remote Repository 644
22.2.9 Working in a New Branch 645
22.2.10 Transferring Changes from a Branch 647
22.3 Summary and Outlook 648
22.3.1 Key Points 648
22.3.2 Recommended Reading 650
22.3.3 Outlook 650

23 Managing Web Projects 651

23.1 Classic Project Management versus Agile Project Management 651
23.1.1 Classic Project Management 652
23.1.2 Agile Project Management 653
23.2 Agile Project Management Based on Scrum 654
23.2.1 The Scrum Workflow 654
23.2.2 The Roles of Scrum 657
23.2.3 Events in Scrum 659
23.2.4 Artifacts in Scrum 663

23.3 Summary and Outlook 665
23.3.1 Key Points 665
23.3.2 Recommended Reading 666
23.3.3 Outlook 666

Appendices 667

A HTTP 669
B HTML Elements 691
C Tools and Command References 703
D Conclusion 715
E The Author 717

Index 719

Index

@import rule 85
& character 440
& operator 430
| character 454

A

a11y 230
AAA phases 534
Abstraction 380
Access control 575
Accessibility 230
  achromatopsia 256
  automated tests 254
  defining the language 251
  deuteranopia 256
  expert tests 254
  forms 239
  headings 239
  images 246
  keyboard shortcuts 252
  keyboard support 252
  linear layout 251
  links 248
  manual tests 254
  protanopia 256
  subtitles 253
  tables 241
  tritanopia 256
  user testing 255
Accessible Rich Internet
  Applications (ARIA) 249
  properties 249
  roles 249
  states 249
Access method 382
Achromatopsia 256
Act phase 534
Adaptive design 327
Addition 143, 433
Adobe Photoshop 198, 613
Adobe Photoshop Elements 198
Affinity Photo 198, 613
Agent 158
Aggregation 383
Agile manifesto 653
Agile project management 653
Agile software development 531
Ajax (Asynchronous JavaScript
  and XML) 218
Algorithm 139
Alpha channel 194
Alternative text 64
Amazon CloudFront 611
Ambient Light API 224
Analysis phase 652
Anchor 63
AND operator 143, 433
Android 320
Angular 288
Animated GIF 194
Anonymous functions 442, 443
Apache Cassandra 524
Apache HBase 524
API gateway 361
appendChild() 214
Apple 320
Application logic 352, 357, 367
Application Programming
  Interface (API) 208
ArangoDB 523
Architecture 351
  client-server 353
  component-based 362
  computer 355
  microservices 361
  monolithic 358
  node 355
  N-tier 353
  P2P 355
  peer-to-peer 355
  service-oriented 359
  two-tier 354
Arithmetic operators 433
Arrange phase 534
Array 140
  iterating 438
  PHP 429
Array-literal notation 152
Array operators 433
Arrow function 148, 443
Artificial intelligence (AI) 391
  application 391
Assertion 532
Assert phase 534
Assignment operator 143, 433

Assistive technologies 230
Association 383
Associative array 519
Asymmetric cryptography 579, 581
Asymmetric encryption 579, 581
Asynchronous communication 218
Atom 43
Attribute 52, 54, 379
Attribute node 209
Attribute selectors 86, 87
Audio format 202
Audio player 202, 203
Authentication 574
  basic authentication 593
  session-based authentication 594
  token-based authentication 595
Authorization 575, 593
AUTOINCREMENT 500
Automated test 527
Automatic builds 44

B

Backend 29
Backend developer 39
Barrier-Free Information Technology Ordinance (BITV) 231
Base64 198
Basic authentication 593
Battery Status API 224
Behavior 379
Bidirectional data binding 370
Binary code 139
Bitmap format 195
Bit operators 433
Bitwise AND 433
Bitwise NOT 433
Bitwise OR 433
Bitwise shift 433
Bitwise XOR 433
Blind users 231
Block cipher 579
Block element 76
Blocking I/O 401
Block-level element 113
Boolean 142
Braille keyboard 231
Branch 140, 144, 645
Breakpoints 331
Broken access control 575
Broken authentication 574
Browser 29, 46
  cache 615
  caching 615
  database 616
  developer tools 43
  Google Chrome 46
  Microsoft Edge 46
  Mozilla Firefox 46
  Opera 46
  Safari 46
  session 171
  test 530
Bug 149, 656
Build 548
Build process 548
Business logic 357, 367
Business logic layer 352
Business owner 657
Button 72
Bytecode 377

C

C 393, 425, 435
C# 393
C++ 393
Cache 520, 611
  browser 615
  client-side 615
  server-side 611
Callback function 402, 509
Canvas API 224
Caption 65
catch 150
Certificate 582
Character string 142
Checkbox 72
Child class 382
Child selector 88
Chocolatey 708
CI server 548
Class 140, 150, 380
Class abstraction 448
Class-based programming language 383
Class constants 446
Class diagram 381
Classic project management 652
Classless programming 383
Class selector 86
Client 29, 158, 218, 353
Client/server protocol 158
Client-server architecture 353

Client-side caching 615
Cloudflare 611
Cloud hosting 554
Code coverage 537
Color names 91
Column-oriented database 523
Command Line API 224
Comma Separated Values (CSV) 182
Commit 632
Common Language Runtime 394
Common methods 449
Comparison operator 433
Compatibility test 530, 708
  tools 708
Compiler 36, 374
Component-based architecture 362
Component test 530, 531, 708
  preparing 534
  running 536
  structure 531
  tools 708
  writing 535
Component with known vulnerabilities 578
Composition 383
Compound data types 429
Concatenation 143
Condition 140, 146
Conditional request 678
confirm() 136
Confirmation dialogue box 136
Conformity levels 235
Console 137
Constant 140, 141
  PHP 432
Constructor function 150
Container virtualization 557
Content delivery networks (CDNs) 609
Content management system (CMS) 391
  Joomla 391
  WordPress 391
Content negotiation 476
Content Security Policy (CSP) 577, 587
Content type 308
Contravariance 449
Controller 367
Control structure 139, 140, 144
  PHP 435
Conversion rate 600
Cookies 170
Core web vitals 604
CouchDB 522
Counter variable 146
Counting loop 146, 438
Coupling 383
Covariance 449
Coverage report 537
Crawler 600
createElement() 214
Create React App 290
  options 290
Cross-browser test 530
Cross-cutting concern 572
Cross-origin requests 584
Cross-Origin Resource Sharing (CORS) 585
  whitelist 586
Cross-site scripting 577
CRUD operation 474, 505
Cryptographic algorithm 579
Cryptography 580
  asymmetric 579, 581
  symmetric 579, 580
CSS 32, 33, 79, 298
  @import rule 85
  adjacent sibling selectors 87
  adjusting the font size 95
  adjusting the font style 95
  attribute selectors 86, 87
  calculating the specificity 90
  centered alignment 97
  child selectors 88
  class selectors 86
  defining fonts 93
  defining the background color 91
  defining the text color 91
  descendant selectors 88
  designing borders 108
  designing ordered lists 104
  designing unordered lists 103
  external 80, 81
  flexbox layout 118
  flex container 118
  flex item 118
  float layout 114
  frameworks 131
  general sibling selectors 87
  grid container 125
  grid item 125
  grid layout 124
  horizontal text alignment 97
  ID selectors 86
  including as external file 81
  inheritance 91
  inline 81, 84
  internal 81, 83

CSS (Cont.)
justification ... 97
layout systems ... 112
left alignment ... 97
letter spacing ... 97
line spacing ... 97
media queries ... 131
notations ... 96
optimizing ... 618
property ... 130
pseudo-classes ... 90, 97
pseudo-elements ... 90
responsive design ... 131
right alignment ... 97
shadow effect ... 97
shorthand property ... 108
sibling elements ... 87
source map files ... 268
specificity ... 89
text decorations ... 96
type selectors ... 85
universal selectors ... 86
value ... 80
vertical text alignment ... 97
viewport ... 114
word spacing ... 97
CSS loader ... 299
CSS post-processors ... 264
CSS preprocessor languages ... 262
implementing custom functions ... 278
using branches ... 271
using functions ... 276
using loops ... 272
using operators ... 270
using variables ... 266
CSS preprocessors ... 261
Less ... 264
Sass ... 264
source map files ... 268
Stylus ... 264
CSS rules ... 33, 50, 80, 130
defining as an attribute ... 84
CSS selectors ... 85
adjacent sibling selectors ... 87
attribute selectors ... 86, 87
child selectors ... 88
class selectors ... 86
descendant selectors ... 88
general sibling selectors ... 87
ID selectors ... 86
type selectors ... 85
universal selectors ... 86
Cumulative layout shifts ... 605
cURL ... 173, 442, 457, 483
Curly brackets ... 435
Cursive font ... 95
CVS ... 632
Cyberduck ... 550

D

Daily scrum ... 655, 661
Daily standups ... 662
Database ... 36, 357, 457
Apache Cassandra ... 524
Apache HBase ... 524
ArangoDB ... 523
attributes ... 496
column-oriented ... 523
CouchDB ... 522
CRUD operations ... 505
data record ... 496
document-oriented ... 520
graph ... 522
key-value ... 519
MariaDB ... 498
Memcached ... 520
MongoDB ... 522
MySQL ... 498
Neo4J ... 523
non-relational ... 37, 518
PostgreSQL ... 498
primary key ... 500
properties ... 496
record ... 496
Redis ... 520
relational ... 37, 496
relations ... 496
SQLite ... 498
tuple ... 496
Data encapsulation ... 380, 381
Data format
CSV ... 182
JSON ... 187
XML ... 182
Data integrity ... 578
Data layer ... 352, 357
Data record ... 496
Data retention ... 367
Data type ... 140
primitive ... 141
return values ... 441
Data URI ... 197
Debugging ... 138

Declaration variables ... 428
Declarative programming ... 378, 385
Decrement operator ... 143, 433
Decryption key ... 579
Dedicated hosting ... 554
Defining the language ... 251
Definition of done ... 662
Deleting a file ... 413
Depended-on components ... 540
Deployment ... 549
Deployment phase ... 652
Descendant selector ... 88
Deserialization ... 577
Designing lists ... 103
Design phase ... 652
Desktop-first ... 328
Destructuring ... 296, 303
Deuteranopia ... 256
Developer console ... 137
Developer tools ... 43, 55
Development dependency ... 409
Development environment ... 44
Development team ... 655, 657, 658
Device Orientation API ... 224
DevOps ... 39
DevOps specialists ... 39
Division ... 143, 433
Django ... 391
DNS lookup ... 624
DNS prefetch ... 624
DNSSEC ... 582
Docker
command reference ... 711
configuring a custom image ... 559
container ... 557
deleting an image ... 713
Dockerfile ... 559
downloading an image ... 713
Hub ... 560
image ... 551, 557
listing images ... 713
network ... 569
registry ... 551
Store ... 560
uploading an image ... 713
volume ... 569
Docker Compose ... 567
command reference ... 713
continuing services ... 714
creating services ... 713
listing services ... 714
pausing services ... 714
Docker Compose (Cont.)
services ... 567
starting services ... 713
stopping services ... 714
Doctype ... 53
Document ... 209
Document node ... 209
Document object ... 144
Document Object Model (DOM) ... 155, 184, 208
API ... 208
DOM tree ... 184, 208
parser ... 184
Document store ... 520
Document type definition (DTD) ... 186, 575, 692
Dollar sign $ ... 428, 443
Domain ... 31, 549
Domain hosting ... 549
Domain Name System (DNS) server ... 32
Domain Name System Security Extensions ... 582
Double underscore ... 432
Drag & Drop API ... 224
Dyschromatopsia ... 231

E

Ecma International ... 134
ECMAScript ... 134
Edge ... 522
Edge server ... 610
Editor ... 43
Element ... 52
Element node ... 209
Encapsulation ... 381
Encryption
asymmetric ... 579, 581
symmetric ... 579, 580
Encryption key ... 579
End-to-end tests ... 530, 708
tools ... 708
Entity header ... 682
Epics ... 656
Equality ... 433
Error exception ... 444
Error message ... 455
Error type ... 443
Escape characters ... 76
Escape code ... 77
European Computer Manufacturers Asssociation (ECMA) ... 134

Event listener 217
Event loop 402
Exclusive or 433
Executable machine code file 375
Exercise phase (unit testing) 533
Exponential operator 143
Express 420, 477
  middleware 420
  processing form data 421
  providing static files 420
  REST 477
Extensible Markup Language (XML) 182
Extreme programming 531

F

Fallback font 93, 94
False 142
Fantasy font 95
Fat arrow function 148
Fetch API 223, 298, 306
Fiber (PHP) 425
Fields 379
File API 224
File Transfer Protocol (FTP) 31, 465
File upload 72
FileZilla 550
First class object 384
First contentful paint (FCP) 603
First input delay 605
First meaningful paint (FMP) 603
First paint (FP) 602
Flexbox layout 118
Flex container 118
Flex item 118
Float layout 114
Font
  cursive 95
  defining 93
  fallback 94
  fantasy font 95
  italic 95
  non-proportional font 94
  sans-serif 94
  serif font 94
ForkLift 550
Form
  buttons 72
  checkboxes 72
  file uploads 72
  password fields 72
  radio buttons 72
Form (Cont.)
  selection lists 72
  text areas 72
  text fields 72
Form data
  receiving 452
  sanitizing 453
  validating 454
  verifying 452
Fragment 31
Frontend 29
Frontend developer 39
Fullscreen API 225
Function 147, 378
  anonymous 147
  body 147
  calling 147
  declaration 147
  defining 147, 439
  defining return values 149
  expression 147
  parameters 148
  pass parameters 148
Functional programming 379, 384
Function parameters 439

G

Gateway 361, 679
General header 682
Geolocation API 225
Getter 382
GIF 193, 613
  animated 194
Gimp 198, 613
Gin 393
Git 635
  adding a remote repository 643
  branch 644, 649
  checkout 649
  clone 649
  cloning 639
  cloning an existing Git repository 639
  command reference 709
  commit 649
  committing changes to the local repository 641
  creating a new Git repository 638
  development branch 644
  fork 649
  git add 640
  git branch 646

Git (Cont.)
  git checkout 647
  git clone 639
  git commit 641
  git fetch 650
  git init 639
  git pull 644
  git push 643
  git remote add 643
  git reset 641
  index 636, 649
  initializing a new Git repository 639
  local working directory 636
  main development branch 645
  master branch 644, 645
  merge 649
  modified 642
  pull 649
  push 649
  remote repository 648
  removing changes from staging area 641
  repository 648
  staged 642
  staging area 636, 649
  states 642
  transferring changes from the remote repository 644
  transferring changes to the staging area 640
  unmodified 642
  untracked 642
  working area 649
  working directory 649
Global namespace 431
Go 393
Google Chrome 46
Graph 522
GraphQL 464, 488
Graph Query Language 489
Grid areas 335
Grid container 125
Grid item 125
Grid layout 124
gzip format 622

H

Hard links 413
Head/eye controls 231
Headless browser 29
Hearing impairments 231
Hex values 91
High-contrast mode 231
High-level programming language 374
High Resolution Time API 225
History API 225, 314
Homebrew 708
Host 549
Hosting 549
  cloud hosting 554
  dedicated hosting 554
  server 549
  shared hosting 552
  types 552
  VPS hosting 553
Host name 31
HSLA value 92
HSL value 92
HTML 32, 51
  abbreviations 56
  acronyms 56
  block-level elements 113
  comments 77
  Data URI 197
  definition lists 58
  headings 56
  highlighting 56
  inline-level elements 113
  line breaks 56
  link 59
  nested lists 58
  ordered lists 57
  paragraphs 56
  quotes 56
  sibling elements 87
  source references 56
  term definitions 56
  texts 56
  unordered lists 57
  version 52
HTML attribute 33, 52, 54
  id 63
HTML element 32, 52
  <a> 59, 695
  <abbr> 56, 695
  <address> 693
  <area> 699
  <article> 77, 238, 693
  <aside> 77, 238, 693
  <audio> 202, 698
  <b> 56, 696
  <base> 692
  <bdi> 696
  <bdo> 697

HTML element (Cont.)
<blockquote> 56, 694
<body> 692
*
* 56
*
* 694, 697
<button> 700
<canvas> 699
<caption> 242, 699
<cite> 56, 695
<code> 695
<col> 699
<colgroup> 699
<datalist> 700
<dd> 58, 694
<del> 697
<details> 694
<dfn> 56, 695
* * 76, 694
<dl> 58, 694
<dt> 58, 694
** 56, 695
<embed> 698
<fieldset> 73, 700
<figcaption> 65, 694
<figure> 65, 694
<footer> 77, 238, 693
<form> 73, 700
<h1> 56, 693
<h2> 56, 693
<h3> 56, 693
<h4> 56, 693
<h5> 56, 693
<h6> 56, 693
<head> 692
<header> 77, 238, 693
<html> 692
<i> 56, 696
<iframe> 698
<img> 64, 698
<input> 700
<ins> 697
<kbd> 696
<label> 73, 700
<legend> 700
<li> 57, 694
<link> 82, 692
<main> 693
<map> 699
<mark> 696
<meta> 692
<meter> 701
<nav> 77, 238, 693

HTML element (Cont.)
<noscript> 136, 701
<object> 698
<ol> 57, 694
<optgroup> 701
<option> 73, 701
<output> 701
<p> 56, 694
<param> 698
<pre> 694
<progress> 701
<q> 56, 695
<rp> 696
<rt> 696
<ruby> 696
<s> 695
<samp> 695
<script> 135, 701
<section> 77, 238, 692
<select> 73, 700
<small> 695
** 76, 697
** 56, 695
<style> 83, 692
<sub> 56, 696
<summary> 694
<sup> 56, 696
<svg> 699
<table> 66, 699
<tbody> 67, 699
<td> 66, 700
<textarea> 73, 701
<tfoot> 67, 700
<th> 66, 700
<thead> 67, 700
<time> 695
<title> 692
<tr> 66, 700
<track> 699
<u> 696
<ul> 57, 694
<var> 695
<video> 199, 698
<wbr> 697
HTTP 31, 158
body 159
conditional requests 678
entity header 160, 161
general header 160, 161
header 159, 162
initial line 159
long polling 174

HTTP (Cont.)
meta information 159
methods 160, 164
payload 159
polling 174
proxy 673
request body 159, 161
request header 159, 160, 682
request line 159
response body 159, 162
response header 159, 161, 686
response line 159
start line 159
status code 161, 166
status line 159
status text 161
verbs 164
HTTP/2 174
HTTP client 29, 158
HTTP handshake 672
HTTP header 159
HTTP method 160, 164
CONNECT 166
DELETE 165
GET 165
HEAD 165
OPTIONS 165
PATCH 165
POST 165
PUT 165
TRACE 165
HTTP status code 669
100 ..... 670, 672
101 ..... 670, 672
200 ..... 166, 670, 672
201 ..... 670, 673
202 ..... 670, 673
203 ..... 670, 673
204 ..... 670, 673
205 ..... 670, 673
206 ..... 670, 673
300 ..... 670, 674
301 ..... 166, 670, 674
302 ..... 670, 674
303 ..... 670, 674
304 ..... 670, 674
305 ..... 670, 674
306 ..... 670, 675
307 ..... 670, 675
308 ..... 670, 675
400 ..... 167, 670, 675
401 ..... 167, 670, 676

HTTP status code (Cont.)
402 ..... 670, 676
403 ..... 167, 671, 676
404 ..... 167, 671, 676
405 ..... 167, 671, 676
406 ..... 671, 676
407 ..... 671, 676
408 ..... 671, 676
409 ..... 671, 677
410 ..... 671, 677
411 ..... 671, 677
412 ..... 671, 677
413 ..... 671, 677
414 ..... 671, 677
415 ..... 671, 677
416 ..... 671, 677
417 ..... 671, 678
418 671
421 ..... 671, 678
426 ..... 671, 678
428 ..... 671, 678
429 ..... 671, 678
431 ..... 671, 678
451 ..... 671, 678
500 ..... 167, 671, 679
501 ..... 671, 679
502 ..... 671, 679
503 ..... 672, 679
504 ..... 672, 679
505 ..... 672, 679
511 ..... 672, 680
Accepted 670, 673
Bad Gateway 671, 679
Bad Request 167, 670, 675
Conflict 671, 677
Continue 670, 672
Created 670, 673
Expectation Failed 671, 678
Forbidden 167, 671, 676
Found (Moved Temporarily) 670, 674
Gateway Timeout 672, 679
Gone 671, 677
HTTP Version Not Supported 672, 679
I'm a Teapot 671
Internal Server Error 167, 671, 679
Length Required 671, 677
Method Not Allowed 167, 671, 676
Misdirected Request 678
Moved Permanently 166, 670, 674
Multiple Choices 670, 674
Network Authentication Required 672, 680

HTTP status code (Cont.)
No Content ... 670, 673
Non-Authoritative Information ... 670, 673
Not Acceptable ... 671, 676
Not Found ... 167, 671, 676
Not Implemented ... 671, 679
Not Modified ... 670, 674
OK ... 166, 670, 672
Partial Content ... 670, 673
Payment Required ... 670, 676
Permanent Redirect ... 670, 675
Precondition Failed ... 671, 677
Precondition Required ... 671, 678
Proxy Authentication Required ... 671, 676
Requested Range Not Satisfiable ... 671, 677
Request Entity Too Large ... 671, 677
Request Header Fields Too Large ... 671, 678
Request Timeout ... 671, 676
Reset Content ... 670, 673
See Other ... 670, 674
Service Unavailable ... 672, 679
Switching Protocols ... 670, 672
Temporary Redirect ... 670, 675
Too Many Requests ... 671, 678
Unauthorized ... 167, 670, 676
Unavailable For Legal Reasons ... 671, 678
Unsupported Media Type ... 671, 677
Upgrade Required ... 671, 678
URI Too Long ... 671, 677
Use Proxy ... 670, 674
HTTP tunnel ... 166
Hue ... 92
Hybrid application ... 323
Hypertext preprocessor ... 425
Hypertext Transfer Protocol
Secure (HTTPS) ... 31, 581, 582

I

Identity ... 433
ID selectors ... 86
Image format
animated GIF ... 193
GIF ... 193
JPEG ... 193
JPG ... 193
PNG ... 194
PNG-24 ... 194
PNG-32 ... 194
PNG-8 ... 194
SVG ... 195
WebP ... 196
Image map ... 699
Image processing ... 198
Adobe Photoshop ... 613
Affinity Photo ... 613
Gimp ... 613
Image processor ... 198
Images
including ... 197
making accessible ... 246
Imperative programming ... 378, 385
Implementation ... 208
Implementation phase ... 652
Including audio files ... 202
Including videos ... 200
Increment expression ... 146
Increment operator ... 143, 433
Index ... 152
IndexedDB ... 616
IndexedDB API ... 225
Indirect input ... 543
Indirect output ... 542
Inequality ... 433
Information hiding ... 381
Inheritance ... 91, 380, 382, 447
Initialization ... 146, 428
Initiatives ... 656
Injection ... 572
Inline element ... 76
Inline-level elements ... 113
Input and output
blocking ... 401
non-blocking ... 402
Input dialog ... 136
Insecure deserialization ... 577
insertBefore() ... 214
Instance ... 380
Insufficient logging and monitoring ... 579
Integrated development
environment (IDE) ... 44
Integrated PHP functions ... 442
Integration test ... 530, 708
tools ... 708
Integrity ... 578
Interactivity ... 604
Intermediate code ... 377
Intermediate language ... 375, 377
Interpreter ... 374
iOS ... 320
iPad ... 320
IP address ... 32
iPhone ... 320
IPv4 ... 32

IPv6 ... 32
Isomorphic JavaScript ... 400
Issues ... 656
Italic font ... 95

J

Java ... 320
JavaScript ... 32, 34, 389
integration in HTML ... 135
JavaScript Object Notation (JSON) ... 187
parser ... 189
schema ... 191
validator ... 192
Jest ... 534
JIT compiler ... 425
Joint Photographic Experts Group ... 193
Joomla ... 391
JPEG ... 193
JPG ... 193, 613
JSON.parse() ... 578
JSON.stringify() ... 578
JSON Web Token (JWT) ... 596
JSX ... 292, 343
logical operations ... 295
loops ... 295
Juggling with data types ... 429

K

Kanban ... 653
Keyboard shortcut ... 252
Keyboard support ... 252
Key-value-store ... 519
Keyword
global ... 432
private ... 446
protected ... 446
public ... 446
static ... 432
Kotlin ... 320

L

LAMP stack ... 37, 391
Largest contentful paint (LCP) ... 605
Late static binding ... 449
Layered architecture ... 352
Layers ... 352, 353
application logic ... 352
presentation logic ... 352
user interface logic ... 352
Layout system ... 112
Lazy loading ... 623
Lean management ... 653
Less ... 264
Linear layout ... 251
Link
external ... 59
internal ... 59, 63
relative ... 59, 61
Link text ... 59
Literal notation ... 151
Living standard ... 52
Load balancing ... 361
Load test ... 531
Load time ... 600
Local storage ... 616
Logical error ... 149
Logical operators ... 433
Logical programming ... 379
Logic layer ... 352, 357
Long polling ... 175
Look-and-feel ... 338
Loop ... 140, 144, 146, 437
head-controlled ... 146
iteration ... 146
tail-controlled ... 146

M

Machine code ... 36, 139, 425
Machine language ... 139
MacPorts ... 708
Magic constants ... 432
Making forms accessible ... 239
Making links accessible ... 248
MAMP ... 426
MAMP stack ... 37
Manual test ... 528
MariaDB ... 498
Markup language ... 32, 35, 182
Masking ... 77
Mathematical functions ... 276
MEAN stack ... 38
Media Capture and Streams ... 225
Media features ... 331
Media queries ... 131, 330
Media rules ... 330
Memcached ... 520, 611
Merge conflicts ... 648
MERN stack ... 38
Message broker ... 364, 365
Message bus ... 359, 365

Message routing 365
Messaging system 364
Metadata 77
Method 379
Metrics 601
  cumulative layout shifts 605
  first contentful paint 603
  first input delay 605
  first meaningful paint 603
  first paint 602
  largest contentful paint 605
  time to first byte 602
  time to interactive 604
Microfrontends architecture 363
Microservice architecture 361
Microsoft Edge 46
Microsoft Visual Studio Code (VS Code) 44
MIME type 167, 475
  application/json 168
  application/pdf 168
  application/xhtml+xml 168
  application/xml 168
  audio/mp4 168
  audio/mpeg 168
  audio/ogg 168
  image/gif 168
  image/jpeg 168
  image/png 168
  image/svg+xml 168
  multipart/form-data 168
  text/css 168
  text/html 169
  text/javascript 169
  text/plain 169
  text/xml 169
  video/mp4 169
  video/mpeg 169
  video/ogg 169
Minifier 618
Mobile application
  hybrid 323
  native 320
Mobile First 328
Mobile web application 321
Mock 543
Model 367
Model-view-controller (MVC) 367
Model-view-presenter (MVP) 370
Model-view-view model (MVVM) 370
Modular programming 379
Module 379
Module test 530, 531
  structure 531
Modulo 433
Modulo operator 143
MongoDB 522
Monolithic architecture 358
Monolithic frontend 362
Mozilla Firefox 46
Multiple branching 145, 436
Multiplication 143, 433
Multi-threaded server 401
Multithreading 425
Multi-tier architecture 356
MV* architecture pattern 367
MySQL 498

N

Namespace 187
Namespace prefix 187
Native application 320, 376
Native apps 338
Navigation Timing API 225
Negation operator 143
Neo4J 523
Network Information API 225
Node 208, 522
Node.js 389
  development dependencies 409
  express 420
  including modules 403
  including packages 403
  installing packages globally 409
  installing packages locally 408
  Node.js Package Manager (npm) 405
  npx 536
  packages 405
  reading files 411
  request queue 402
  worker 402
  writing files 412
Node.js API 400
Node.js installation
  Linux binary package 707
  Linux package manager 708
  macOS binary package 707
  macOS installation file 703
  Raspberry Pi 708
  Windows binary package 707
  Windows installation file 706
Node.js Package Manager (npm) 405, 705
  creating 290

Node.js Package Manager (npm) (Cont.)
  downloads 289
  i 408
  init 405
  install 408
  Package Runner 705
Node type 209
Non-blocking I/O 402
Non-proportional font 94
NoSQL 37
npmtrends 289
npx 536
N-tier architecture 353, 356
Null coalescing operator 434, 457, 461
Number 141
Numeric data 141

O

Obfuscation 621
Object 140, 379, 429
  cloning 449
  first class 384
Object-based programming language 383
Object instance 380
Objective-C 320
Object-literal notation 151
Object model 445
Object orientation
  basic principles 379
  class-based 382
  prototype-based 382
Object-oriented programming 379
Object-relational mapping (ORM) 516
  library 574
  variants 516
Opera 46
Operator 140, 143
  arithmetic 143
Optimizing connection times 609
Optimizing HTML 618
Optimizing images 612
Optimizing JavaScript 618
Origin 584
OR operator 143, 433
Overload 449
OWASP top ten 572
  broken access control 575
  broken authentication 574
  components with known vulnerabilities 578
  cross-site scripting 577
OWASP top ten (Cont.)
  injection 572
  insecure deserialization 577
  insufficient logging and monitoring 579
  security misconfiguration 576
  sensitive data exposure 574
  XML external entities 575

P

package.json 407
  properties 407
PageSpeed 600
Page Visibility API 225
Parameterized request 574
Parent class 382
Password field 72
Path 31
Peer-to-peer (P2P) architecture 355
Performance
  core web vitals 604
  crawler 600
  CSS 618
  cumulative layout shifts 605
  first contentful paint 603
  first input delay 605
  first meaningful paint 603
  first paint 602
  HTML 618
  images 612
  interactivity 604
  JavaScript 618
  largest contentful paint 605
  load time 600
  metrics 601
  optimizing connection times 609
  PageSpeed 600
  time to first byte 602
  time to interactive 604
  visual stability 604
Performance metrics 601
Performance test 531, 708
  tools 708
Performance Timeline 225
Perl 425
Persistence layer 352, 357
PHP 390
  areas 427
  creating an object 445
  dynamic web pages with 450
  functions 439
  interpreter 390

PHP (Cont.)
local installation of ... 425
operators ... 432
overview of ... 460
sending email with ... 458
writing classes ... 445
Physical limitations ... 231
Platform ... 36
PNG-24 method ... 194
PNG-32 method ... 194
PNG-8 method ... 194
Pointer Events ... 225
Point-to-point communication ... 364
Polling ... 174
Polymorphism ... 380, 383
Port ... 31
Portable Network Graphics (PNG) format ... 194, 613
PostgreSQL ... 498
Post-processors ... 264
Preconnect ... 624
Predefined constants ... 432
Predefined variables ... 430
Predicate programming ... 379
Prefetch ... 625
Prepared statement ... 574
Preprocessors ... 262
Prerender ... 626
Presentation ... 367
Presentation API ... 225
Presentation layer ... 352, 356
Presenter ... 370
Pretty Good Privacy (PGP) ... 582
preventDefault() ... 309
Primary key ... 500
Primitive data types ... 429, 445
Private key ... 581
Procedural programming ... 378
Procedure ... 379
Process model ... 652
V model ... 652
V model XT ... 652
waterfall model ... 652
Product backlog ... 654, 663
Product increment ... 655, 665
Product owner ... 654, 657
Program ... 139
image processing ... 198
native ... 376
Program execution operator ... 433
Programming ... 139
arrays ... 140
branches ... 140
classes ... 140
classless ... 383
conditions ... 140
constants ... 140
control structures ... 140
data type ... 140
declarative ... 378
functional ... 379
imperative ... 378, 385
logic ... 379
logical ... 379
modular ... 379
object-oriented ... 379
objects ... 140
operators ... 140
procedural ... 378
prototypical ... 383
structured ... 378
variables ... 139
Programming language ... 32, 35, 139
C ... 393
C# ... 393
C++ ... 393
class-based ... 383
compiled ... 375
functional ... 384
Go ... 393
higher-level ... 139
interpreted ... 375
Java ... 320
JavaScript ... 389
Kotlin ... 320
object-based ... 383
object-oriented ... 379
PHP ... 390
popularity of ... 387
Python ... 391
Ruby ... 391
TypeScript ... 389
Programming paradigm ... 378
Programming structure ... 427
Programming web applications ... 425
Progress Events API ... 226
Project management
agile ... 653
bugs ... 656
classic ... 652
epics ... 656
initiatives ... 656
issues ... 656

Project management (Cont.)
Kanban ... 653
Lean ... 653
Lean Management ... 653
scenarios ... 655
Scrum ... 654
stories ... 655
subtasks ... 656
tasks ... 656
themes ... 656
use cases ... 655
Promise ... 223, 298, 509
prompt() ... 136
Property ... 130, 379
Protanopia ... 256
Protocol ... 31, 354
Prototype ... 380, 383
Prototype programming ... 383
Proximity API ... 226
Proxy ... 165, 166, 679
Proxy server ... 166
Pseudo-class ... 90, 97, 109
Pseudo-element ... 90
PYPL index ... 387
Python ... 391

Q

Query string ... 31
parameters ... 31

R

Radio button ... 72
Rasmus Lerdorf ... 425
Raspberry Pi ... 708
React ... 287, 289
build process ... 292
child component ... 303
class components ... 293
className ... 301
component hierarchy ... 302
Context API ... 303, 310
controlled components ... 307
createContext ... 310
CSS import ... 299
CSS modules ... 302
data sovereignty ... 304
dependencies ... 298
dumb components ... 293
entry file ... 291
export default ... 295
forms ... 307
fragment ... 299
function components ... 293
Hook API ... 293
htmlFor ... 301
inline styling ... 298
key ... 295
lifecycle ... 296
onchange ... 309
onClick ... 306
props ... 303
provider ... 311
reconciler ... 292
renderer ... 292
rendering ... 296
routing ... 314
setup ... 290
side effects ... 297
smart components ... 293
start ... 291
state ... 295
StrictMode ... 293
style ... 298
styled components ... 302
styling ... 298
TypeScript ... 290
uncontrolled components ... 307
useContext ... 312
useEffect ... 297
useState ... 296
React Native ... 338
React Router ... 314
BrowserRouter ... 314
default route ... 315
link ... 315
navigate ... 315
Record ... 496
Redis ... 520, 611
RedMonk index ... 386
Relational database ... 496
Remote repository ... 634
committing changes ... 643
removeChild() ... 215
Render ... 29
Rendering engine ... 46
Repetition ... 144
replaceChild() ... 215
Repository ... 632
local ... 633
remote repository ... 634
Request body ... 161

Request header 682
Accept 162, 683
Accept-Charset 162, 683
Accept-Encoding 162, 683
Accept-Language 162, 683
Authorization 162, 683
Cache-Control 683
Connection 683
Content-Length 162, 683
Content-MD5 683
Content-Type 162, 683
Cookie 162, 683
Date 162, 683
Expect 684
Forwarded 684
From 684
Host 162, 684
If-Match 684
If-Modified-Since 684
If-None-Match 684
If-Range 684
If-Unmodified-Since 685
Max-Forwards 685
Pragma 685
Proxy-Authorization 685
Range 685
Referer 685
TE 685
Transfer-Encoding 685
Upgrade 685
User-Agent 163, 686
Via 686
Warning 686
Request line 160
Requirements 652
Resource 31
Resource (data type) 429
Resource Timing API 226
Response body 162
Response header 682
Accept-Ranges 686
Age 686
Allow 163, 686
Cache-Control 686
Connection 687
Content-Encoding 687
Content-Language 163, 687
Content-Length 163, 687
Content-Location 163, 687
Content-MD5 687
Content Range 687
Content-Security-Policy 687
Response header (Cont.)
Content-Type 163, 687
Date 163, 687
ETag 687
Expires 687
Last-Modified 163, 688
Link 688
Location 688
Pragma 688
Proxy-Authenticate 688
Retry-After 688
Server 163, 688
Set-Cookie 163, 688
Trailer 689
Transfer-Encoding 689
Vary 689
Via 689
Warning 689
WWW-Authenticate 163, 689
Response line 161
Responsive design 131, 233, 321, 322, 326
Responsive web design 322
REST 366, 464, 471
calling the DELETE route 487
calling the GET route 484, 485
calling the POST route 484
calling the PUT route 486
content negotiation 476
express 477
idempotent requests 475
implementing the DELETE route 482
implementing the GET route 481
implementing the POST route 480
implementing the PUT route 481
principles 472, 473, 475, 476
secure requests 475
workflow 471
REST API 165, 166
calling 483
implementing 476
Return values 149, 441
Revel 393
RGBA value 91
RGB value 91
Rich Internet Application 30
Role-based access control (RBAC) 576
Root node 209
Routine 378
Routing engine 368, 419
Ruby 391
Ruby on Rails 392

Runtime environment 376
Runtime error 149

S

Safari 46
Same-origin policy (SOP) 584
Sanitization filter 453
Sass 264
@each 272
@for 273
@while 275
color functions 276
compiling 265
implementing custom functions 278
inheritance 282
installation 264
list functions 276
map functions 277
mixins 282
modules 276
module system 276
nesting 281
nesting rules 281
selector functions 277
string functions 276
using branches 271
using functions 276
using loops 272
using operators 270
using variables 266
Sass preprocessor 302
Saturation 92
Scalable Vector Graphics (SVG) format 195, 613, 699
Scalar data types 429
Scenarios 655
Schema 31
Screen magnifiers 231
Screen Orientation API 226
Screen reader 29, 231
Scripting languages 377
Scrum 654, 658
artifacts 654
business owner 657
daily scrum 655, 661
daily scrum meeting 655
definition of done 662
development team 655, 657, 658
events 654
product backlog 654, 663
product increment 655, 665
Scrum (Cont.)
product owner 654, 657
product vision 654, 657
roles 654
scrum board 660
scrum master 655, 657, 658
scrum team 655, 657
sprint 655, 660
sprint backlog 655, 664
sprint planning 655, 659
sprint planning meeting 655
sprint retrospective 655, 662
sprint review 655, 662
stakeholder 657
story points 663
user stories 655
Vegas Rule 663
workflow 654
Search engine bot 165
Search engine optimization (SEO) 77, 230, 600
Search engine ranking 600
Section 508 231
Secure File Transfer Protocol (SFTP) 550
Secure Shell (SSH) 582
Secure Sockets Layer (SSL) 171, 410, 581, 582
certificate 549
connection 601
Security Assertion Markup Language (SAML) token 596
Security misconfiguration 576
Security test 531, 709
tools 709
Selection list 72
Selectors 85
adjacent sibling selectors 87
attribute selectors 86, 87
child selectors 88
class selectors 86
descendant selectors 88
general sibling selectors 87
ID selectors 86
type selectors 85
universal selectors 86
Sensitive data exposure 574
Serialization 577
Server 158, 218, 353
Server-sent events (SSEs) 175, 226
Server-side cache 611
Server-side PHP 460
Serverside rendering 291
Service consumer 466

Service discovery 465
Service layer 359
Service-oriented architecture (SOA) 359
Service provider 466
Service registry 465, 466
Session 594
Session-based authentication 594
Session handling 574
Session ID 594
Session tokens 574
Setter 382
Setup phase (unit testing) 533
Shared hosting 552
Shorthand property 108
Sibling elements 87
Side effect 384
Simple API for XML (SAX) 184
Simple Mail Transfer
  Protocol (SMTP) 31, 465
Simple Object Access
  Protocol (SOAP) 366, 464, 465
  body 469
  envelope 469
  fault 469
  header 469
  service consumer 466
  service provider 466
  service registry 466
  workflow 466
Simple Web Token (SWT) 596
Single-page applications 288, 357, 369
Single-threaded server 402
Smartphone 320
Software 139
Software architect 351
Software architecture 652
Software Development Kit (SDK) 320
Software stack 36
Solution stack 36
Source code 36, 374
Source management system 44
Source map files 268
Spaceship comparison operator 434
Spaceship operator 434
Special characters 77
Special data types 429
Specification sheet 652
Specificity 89, 90
Spies (unit testing) 541
Spread operator 311
Sprint 655, 660
Sprint backlog 655, 664
Sprint planning 655, 659
Sprint planning meeting 655
Sprint retrospective 655, 662
Sprint review 655, 662
SQL 498
  AUTOINCREMENT 500
  CREATE TABLE 499
  creating new tables in a database 499
  DELETE 504
  deleting records 504
  IF NOT EXISTS 499
  injection 573
  INSERT 500
  ORDER BY 503
  ORM libraries 574
  parameterized queries 574
  prepared statements 574
  PRIMARY KEY 500
  reading all records of a table 501
  reading records based on specific criteria 503
  SELECT 501
  SQL injection 573
  storing new records in a table 500
  UPDATE 504
  updating records 504
  WHERE 503
SQL dialect 498
SQL injection 511, 573
SQLite 498
Stack 36
Stakeholder 657
Standard ports 31
State 379
Stateless protocol 170
Statement groups 435
Static typing 389
Status code class
  Client Error 675
  Information 672
  Redirection 674
  Server Error 679
  Successful 672
Stories 655
Story points 663
Stress test 531
String 140
String operations 433
Structured programming 378
Stubs 543
Style language 32, 35
Stylesheets 79

Stylus 264
Subclass 382
Subdomain 31
Sublime Text 43
Subresource 625
Subroutine 379
Subtasks 656
Subtitles 231, 253
Subtraction 143, 433
Subversion (SVN) 632
Success criteria 235
Superclass 382
Swift 320
Symbolic links 413
Sym links 413
Symmetric cryptography 579, 580
Symmetric encryption 579, 580
Symmetric key 582
Syntax error 149
System under test 533

T

Table 66, 496
  body 67
  cell 66
  column 66
  description 242
  footer 67, 242
  header 67
  making accessible 241
  row 66
  table body 242
  table header 242
Table headings 242
  horizontal 244
  vertical 244
Tablet 320
Tag
  closing 53
  opening 53
Tasks 656
TCP handshake 601
Teardown phase (unit testing) 533
Technical specification 652
Template engine 369
  client-side 370
Template string 142, 152
Ternary operator 434
Test case 531
Test context 533
Test coverage 537
  determining 538
Test double 540
Test-driven development (TDD) 531
Test fixture 533
Test framework 44
Testing environment 530
Testing phase 652
Testing tools 708
Test pyramid 529
Tests
  AAA phases 534
  act 534
  act phase 534
  advantages 528
  arrange 534
  arrange phase 534
  assert 534
  assert phase 534
  browser tests 530
  code coverage 537
  compatibility tests 530
  component tests 530
  coverage report 537
  cross-browser tests 530
  E2E tests 530
  end-to-end tests 530
  functional tests 530
  integration tests 530
  load tests 531
  manual 528
  mock objects 543
  mocks 543
  module tests 530
  performance tests 531
  security tests 531
  spies 541
  stubs 543
  suite 531
  test coverage 537
  test-driven development 531
  test suite 531
  unit tests 530
Text area 72
Text field 72
Text node 210
Themes 656
Thick client 357
Thin client 357
Three-way operator 434
Tier 353
Time to first byte 602

Time to interactive 604
Time to live (TTL) 611
TIOBE index 385
Token-based authentication 595
Tools 42
  Adobe Illustrator 199
  Adobe Photoshop 198
  Adobe Photoshop Elements 198
  Affinity Photo 198, 199
  curl 173
  Cyberduck 550
  developer tools 55
  FileZilla 550
  ForkLift 550
  Gimp 198
  Inkscape 199
Top-level domain 31
Touch Events 226
Transport Layer Security (TLS) 410, 581, 582
Tritanopia 256
True 142
try 150
Tuple 496
Two-layer architecture 354
Two-tier architecture 353
Type declarations 425, 441
Type juggling 429
Type operator 433
TypeScript 389
Type selector 85
Types of neutralization 453
Typing
  static 389
  weak 389

U

UglifyJS 618
Unified Modeling Language (UML) 381
Uniform Resource Identifiers (URIs) 472
Unit test 530, 531
  preparing 534
  structure 531
  writing 535
Universal Description, Discovery and Integration (UDDI) 465
Universal JavaScript 400
Universal selector 86
URL 29, 30
  absolute 61
  domain 31
  fragment 31
URL (Cont.)
  hostname 31
  path 31
  port 31
  protocol 31
  query string 31
  relative 61
  schema 31
  structure 30
  subdomain 31
  top-level domain 31
Usability 61, 528
Usage 139
Use cases 655
User agent 29, 158
User experience (UX) 600
User interface logic 352
Users, blind 231
User stories 655
User test 528
User Timing API 226

V

Value assignment 140
Variable 139, 140, 428
  reference 430
  scope 432
  type 428
Variable declaration 140, 429
Variable function 443
Variable initialization 140
Variable variables 432
Vector graphic 195
Vegas Rule 663
Verify phase (unit testing) 533
Version control system 44, 631
  central 632
  decentralized 632, 633
Vibration API 226
Video formats 199
Video player 201
Videoplayer 199
Videos
  subtitles 253
View 367
Viewmodel 370
Viewport 114, 328
View template 369
Virtual private server (VPS) hosting 553
Visibility 446
Visual C# 320

Visual impairments 231
Visual stability 604
V model 652
V model XT 652
Vue 289
Vulnerability 572
  broken access control 575
  broken authentication 574
  components with known vulnerabilities 578
  cross-site scripting 577
  injection 572
  insecure deserialization 577
  insufficient logging and monitoring 579
  security misconfiguration 576
  sensitive data exposure 574
  XML external entities 575

W

WAI 231
WAMP 426
WAMP stack 37
Watch mode 265
Waterfall model 652
WCAG 231
  guidelines 233
  principles 232
Weak typing 389
Web accessibility 230
  WAI 231
  WCAG 231
Web Accessibility Initiative 231
Web Animation API 226
Web APIs
  Ambient Light API 224
  Battery Status API 224
  Canvas API 224
  Command Line API 224
  Device Orientation API 224
  Drag & Drop API 224
  File API 224
  Fullscreen API 225
  Geolocation API 225
  High Resolution Time API 225
  History API 225
  IndexedDB API 225
  Media Capture and Streams 225
  Navigation Timing API 225
  Network Information API 225
  Page Visibility API 225
  Performance Timeline 225
Web APIs (Cont.)
  Pointer Events 225
  Presentation API 225
  Progress Events 226
  Proximity API 226
  Resource Timing API 226
  Server-Sent Events 226
  Touch Events 226
  User Timing API 226
  Vibration API 226
  Web Animation API 226
  Web Cryptography API 226
  Web Notification API 226
  Web Speech API 226
  Web Storage API 227
  Web Worker API 227
Web application 29, 30
Web browser 29, 46
Web client 29, 354
Web colors 91
Web components 362
Web Content Accessibility Guidelines 231
Web Cryptography API 226
Web framework
  Django 391
  Gin 393
  Revel 393
  Ruby on Rails 392
Web Hypertext Application Technology Working Group (WHATWG) 52
Web Notification API 226
Webpack 299
Web page 29, 30
  semantics 52
  structure 52
Web performance
  CLS 605
  core web vitals 604
  crawler 600
  crawling 600
  CSS 618
  cumulative layout shifts 605
  first contentful paint 603
  first input delay 605
  first meaningful paint 603
  first paint 602
  HTML 618
  images 612
  interactivity 604
  JavaScript 618
  largest contentful paint 605
  load time 600

Web performance (Cont.)
optimizing connection times 609
PageSpeed 600
time to first byte 602
time to interactive 604
visual stability 604
WebP format 613
Web server 29, 354
routing engine 419
Web service 366, 463
GraphQL 488
REST 471
SOAP 465
Webservices Description Language (WSDL) 465, 467
Website 30
body 53
dynamic 36
header section 53
rendering 29
static 35
Website accessibility 230
WebSocket API 176
WebSocket client 176
WebSocket connection 176
WebSocket protocol 176
WebSocket server 176
Web space hosting 549
Web Speech API 226
Web Storage API 227
WebStorm 44
Web view 323
Web Worker API 227
Whitelist 586
Widget 338, 362
Windows Mobile 320
WordPress 391
Worker threads 411
Work steps 139
World Wide Web Consortium (W3C) 52

X

XAMPP 426
XML 182
attributes 182
document 183
elements 182
external entities 575
namespace 187
schema 186
validator 186
XML parser 184
XSS 577, 587
XXE attack 575

Y

Yarn 290

Philip Ackermann is the CTO of Cedalo GmbH and the author of several reference books and technical articles on Java, JavaScript, and web development. His focus is on the design and development of Node.js projects in the areas of Industry 4.0 and Internet of Things.

Philip Ackermann

Full Stack Web Development

The Comprehensive Guide

740 pages | 08/2023 | $59.95 | ISBN 978-1-4932-2437-1

www.rheinwerk-computing.com/5704