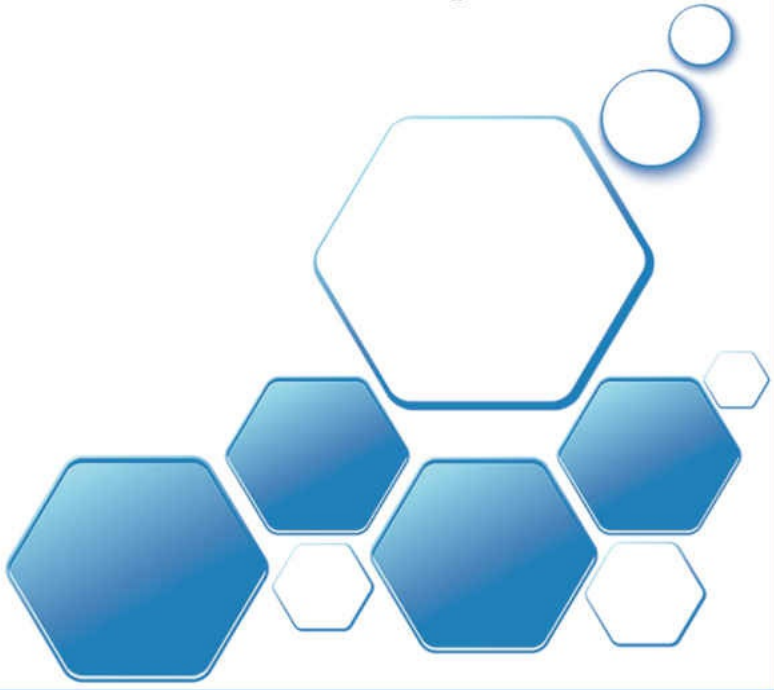
Web Development Crash Course

HTML5 and CSS3

Learn HTML and CSS from Experts



Neo D. Truman

Web Development Crash Course

HTML5 and CSS3
Learn HTML and CSS from Experts

Neo D. Truman

Copyright

Table of Contents

Preface

Book Audience

The book is aimed at web developers and designers who want to learn HTML5 and CSS3.

- It is for beginners in web development and web design.
- It is a complete reference book for those who want to learn HTML and CSS codes for front-end web development.
- For individuals who want to build websites or learn web design.

How to Contact Us

Please email *contact@neodtruman.com* to ask questions about this book.

Follow us on the Amazon Author page, *https://www.amazon.com/Neo-D-Truman/e/B09P6LHL2M*, for news and information about our books.

Acknowledgments

First, I must thank my parents for their love and guidance. Thank you, parents, for buying me my first computer, which started my career path.

Then, I want to express this special thanks to my dear wife. My wife takes care of everything in the family. Thanks to that, I have time to contribute to the community and complete this book.

Finally, my sincere thanks are to my loved ones in my family, friends, colleagues, and people who have supported and helped me.

Conventions Used in This Book

The following conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Bold

Indicates important concepts or UI items, such as menu items and buttons to be selected or clicked.

Constant width

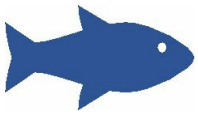
Indicates computer code, including statements, functions, classes, objects, methods, properties, etc.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or values determined by context.

 This element signifies a general note.

 This element signifies a tip or suggestion.

 This element indicates a warning or caution.

Setting Up a Development Environment

The first step in setting up your development environment is choosing your code editor. Almost editors have essential features like autocompletion, syntax highlighting, formatting code, and intelligent refactoring. Using them helps you to prevent potential mistakes. Other features, such as line-by-line debugging, could be already available in the editor. However, we will use web browsers' Developer Tools while debugging JavaScript code. After setting up the development environment, your coding will be effortless and fast.

Several tools are available so that you can choose any of them. However, no matter what tools you choose, you'll follow a similar process to learn the lessons.

Install a Code Editor

This book uses ***Visual Studio Code*** (shortened to VS Code) as the code editor. You can download this free at

https://code.visualstudio.com/download

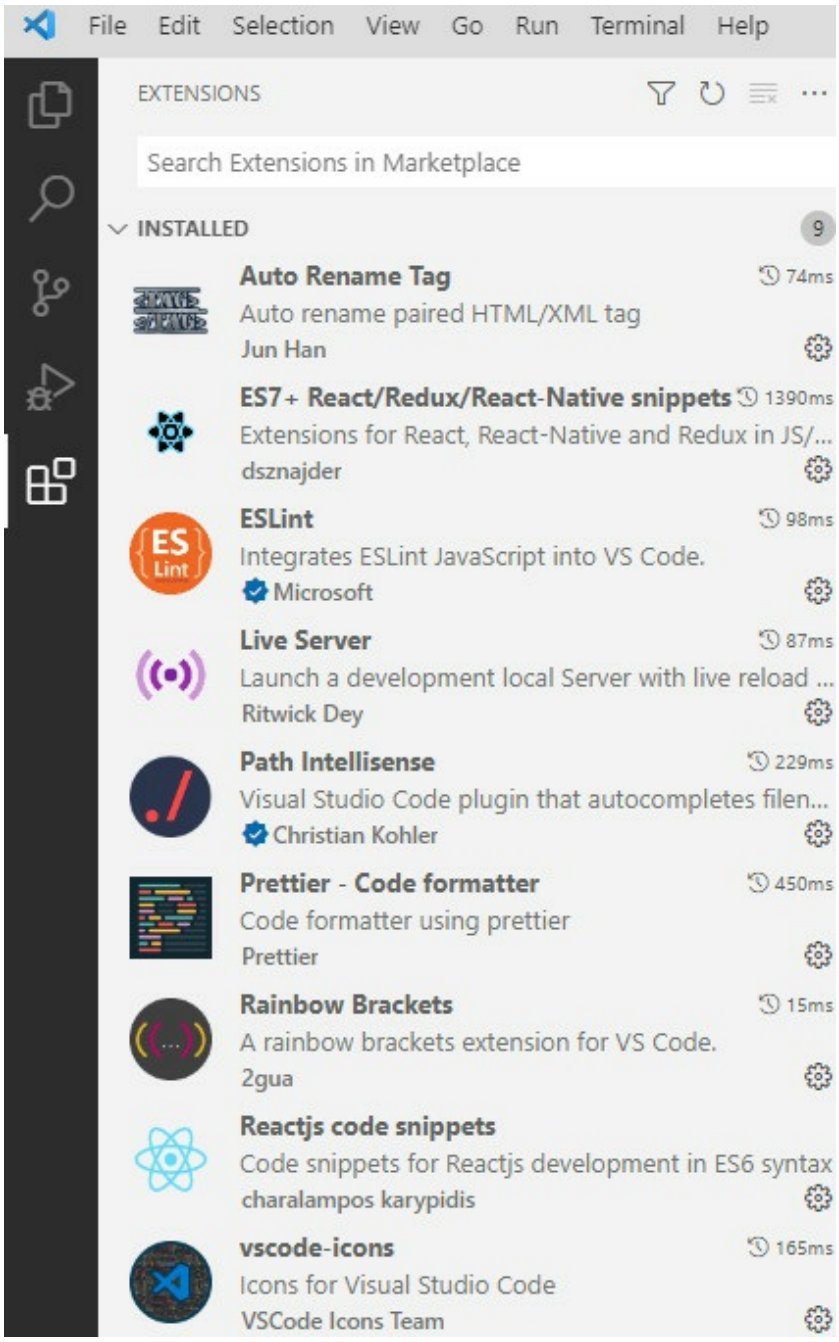
In Stack Overflow's survey, VS Code was ranked as the most popular code editor across languages.

1.1 Install Essential Extensions

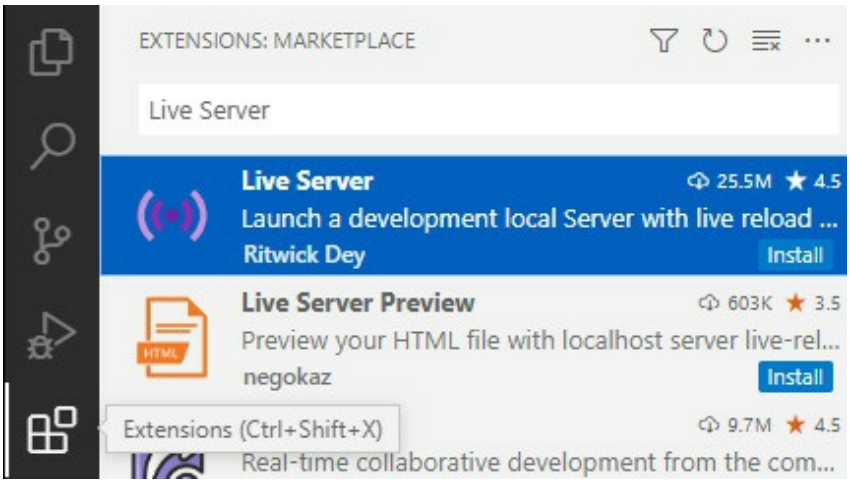
VS Code supports adding extensions that help us code faster and easier. After installing the editor, let's add these essential extensions:

- Prettier - Code formatter: Enforces a consistent style by parsing your code and re-printing it with its rules that take the maximum line length into account, wrapping code when necessary.
- Path Intellisense: Visual Studio Code plugin that autocompletes filenames and paths.
- Auto Rename Tag: Auto rename paired HTML/XML tags.
- ESLint: Finds and fixes problems in your JavaScript code.
- Rainbow Brackets: Provides rainbow colors for the round, square, and squiggly brackets.
- vscode-icons: File and folder icons for Visual Studio Code.

Below is a screenshot of the extensions for your reference when searching for them.



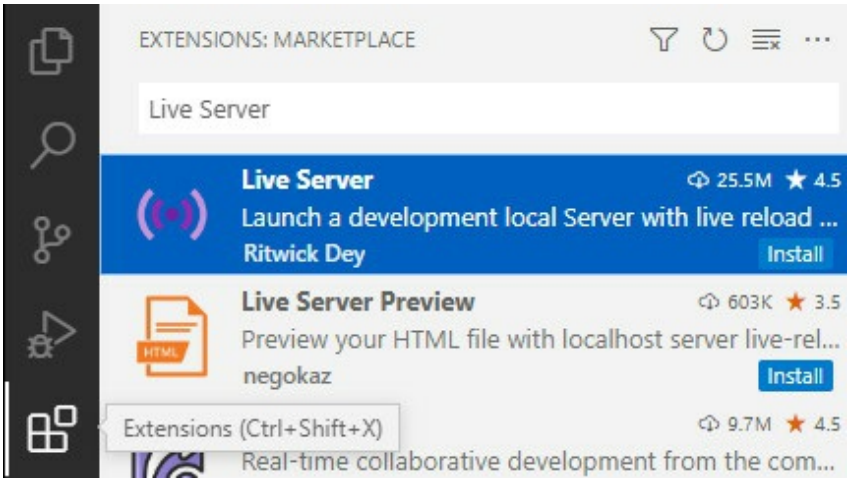
To install the extensions, click on the **Extensions** icon (as in the picture below), then type a name into the **Search Box**; select the extension you want and click its **Install** button.



1.2 Install Live Server

Live Server helps us launch a local development server with a live reload feature for static or dynamic web pages.

To install the extension, click on the **Extensions** icon, then type "Live Server" into the **Search Box**; select the extension in the picture below and click its **Install** button.



How to run an HTML file with the Live Server?

- Step 1: Open your HTML file with the Visual Studio Code.
- Step 2: Right-click on the editing area, select "**Open with Live Server**," or click the "**Go Live**" button at the bottom-right corner of the VS Code.

After doing the above steps, your HTML file will be loaded into a live

server and available at this address:

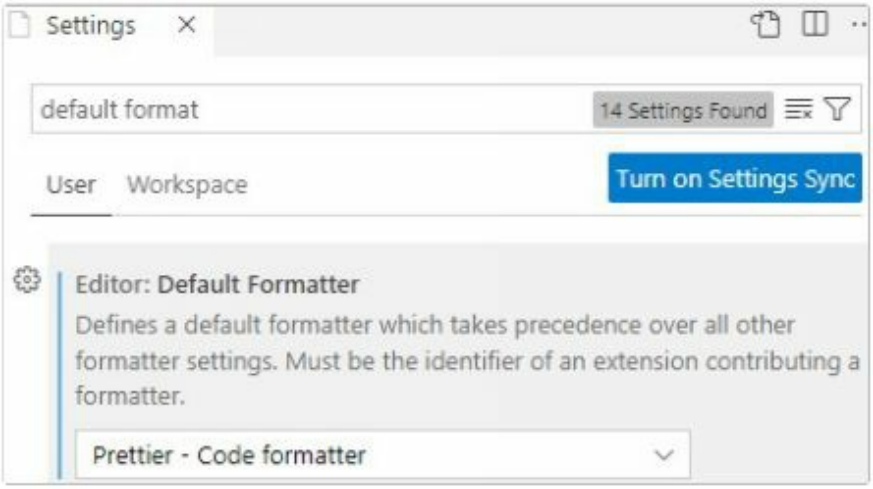
http://127.0.0.1:5500/

You must manually refresh your web page only once after waking up your PC.

1.3 Setting Default Formatter

We should configure the Default Formatter to use the “Prettier - Code formatter” extension. Please follow these steps to turn it on:

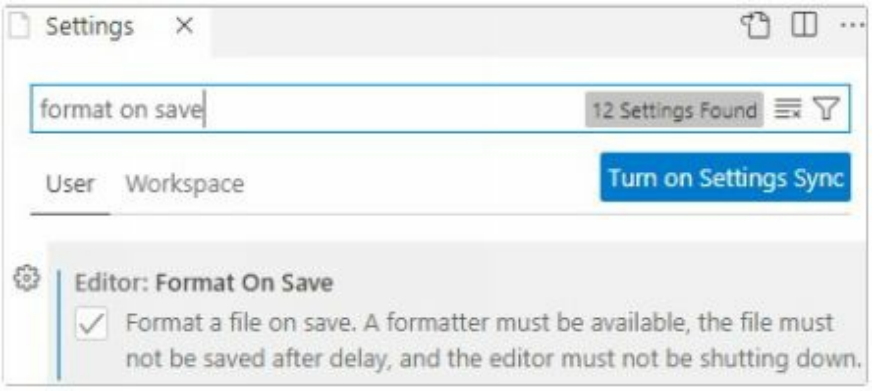
- Go to the menu and select: **File > Preferences > Settings**
- Type “default format” into the **Search Box**
- Then select the “**Prettier - Code formatter**” option as in the below picture.



1.4 Enable Formatting Code on Save

We should enable the “Format on Save” feature to have the best readable code. Please follow these steps to turn it on:

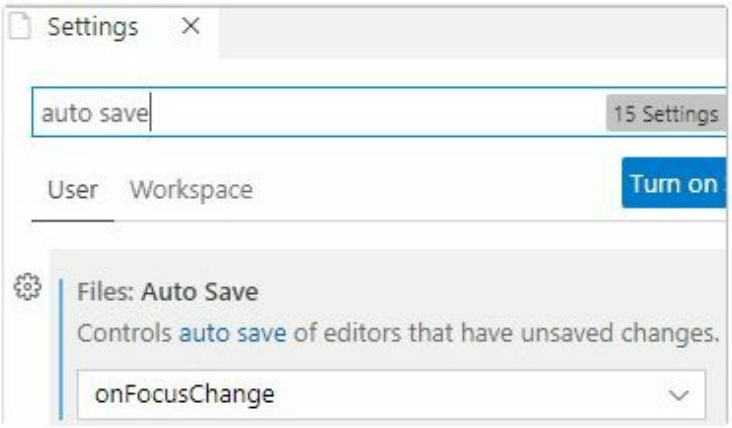
- Go to the menu and select: **File > Preferences > Settings**
- Type “format on save” into the **Search Box**
- Then check the “**Editor: Format On Save**” checkbox as in the picture below.



1.5 Auto Save Files on Focus Change

We should enable the “File: Auto Save” feature on focus change. Please follow these steps to turn it on:

- Go to the menu and select: **File > Preferences > Settings**
- Type “auto save” into the **Search Box**
- Then select the “**onFocusChange**” option as in the below picture.

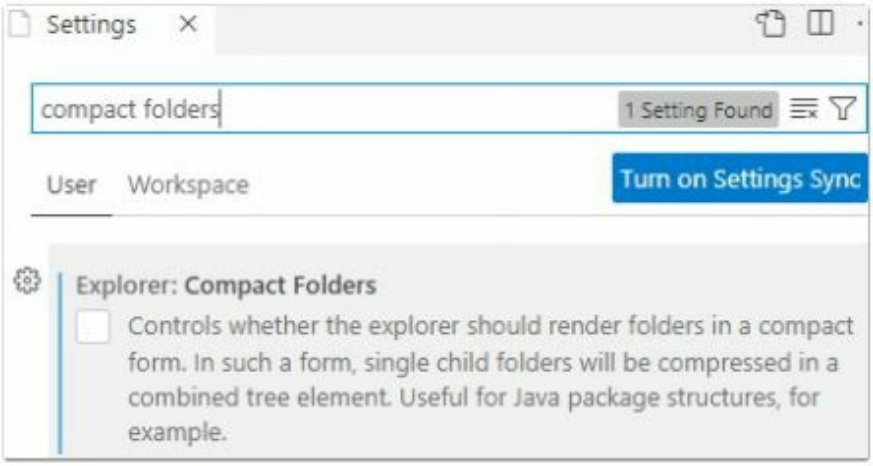


1.6 Disable Compact Folders

VS Code renders folders in its explorer in a compact form. In such a form, single child folders will be compressed in a combined tree element. It’s only helpful for Java package structures. Therefore, we should disable the “Compact Folders” feature by following these steps:

- Go to the menu and select: **File > Preferences > Settings**
- Type “compact folders” into the **Search Box**

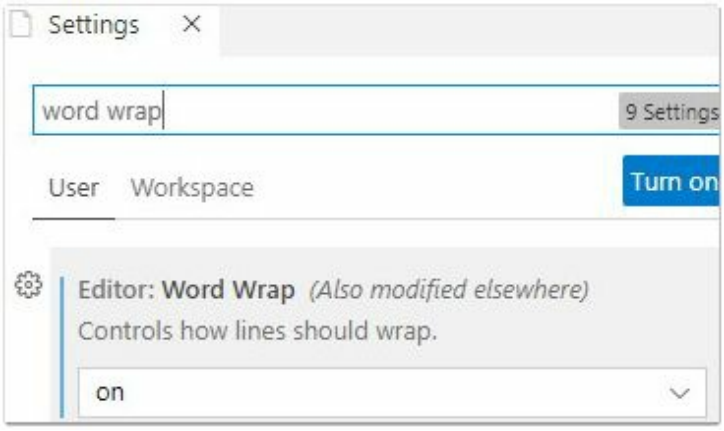
- Then uncheck the "**Explorer: Compact Folders**" checkbox, as in the picture below.



1.7 Enable Word Wrap

To get rid of the horizontal scroll bar, we could turn on the "Word Wrap" feature by following these steps:

- Go to the menu and select: **File > Preferences > Settings**
- Type "word wrap" into the **Search Box**
- Then select option "**on**" of "**Editor: Word Wrap**" as in the below picture.

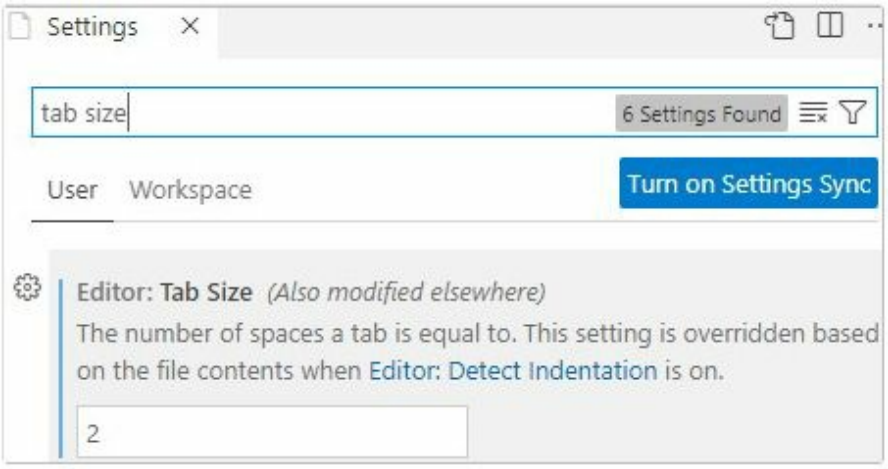


1.8 Set Tab Size Smaller

The default number of spaces in a tab is 4. But according to my experience, we need only two spaces. If you want to change the default

setting like me, then follow these steps:

- Go to the menu and select: **File > Preferences > Settings**
- Type “tab size” into the **Search Box**
- Then set the “**Editor: Tab Size**” number to 2, as shown in the picture below.



2 Using a Web Browser

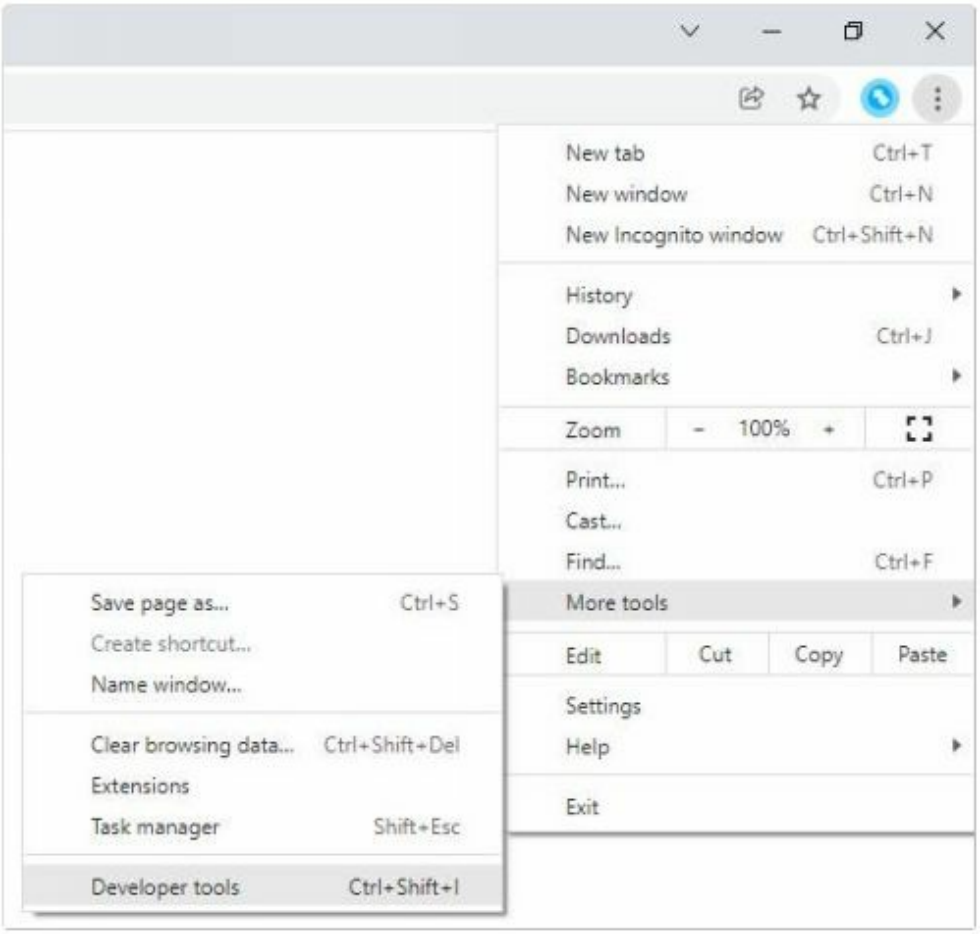
We will use ***Google Chrome*** to run and debug our JavaScript code since it is the most popular browser. Moreover, it also includes fantastic *developer tools* that will be described below.

Anyway, you could choose any browser as your favorite one and follow a similar process.

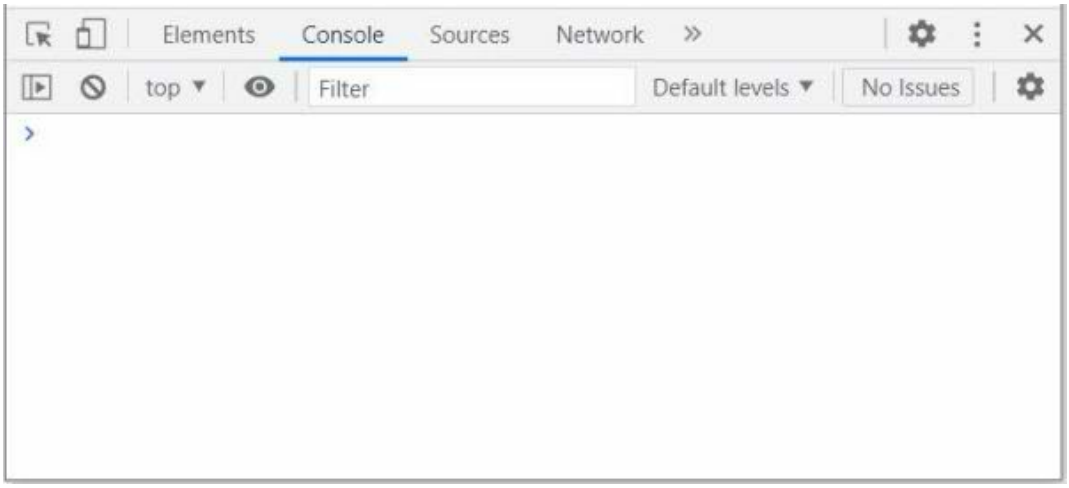
2.1 Using Developer Tools in The Browser

The developer tools are usually presented as a tabbed group of panes at the right or bottom of the web browser window. For example, to open the developer tools in Google Chrome, open the **Chrome Menu** in the upper-right-hand corner of the browser window and select **More Tools** > **Developer tools**.

You can also use **Shift + CTRL + J** (on Windows/Linux) or **Option** + ⌘ **+ J** (on macOS) to open the tools.

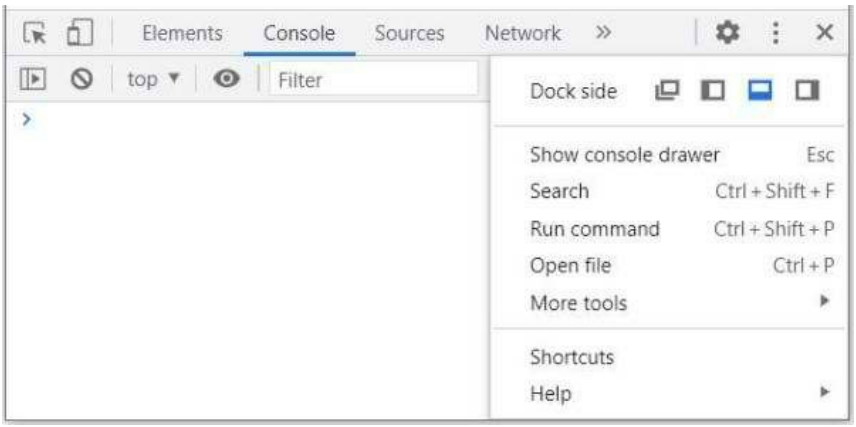


The developer tools will either open up in a new window or within the Chrome window, as in the picture below.



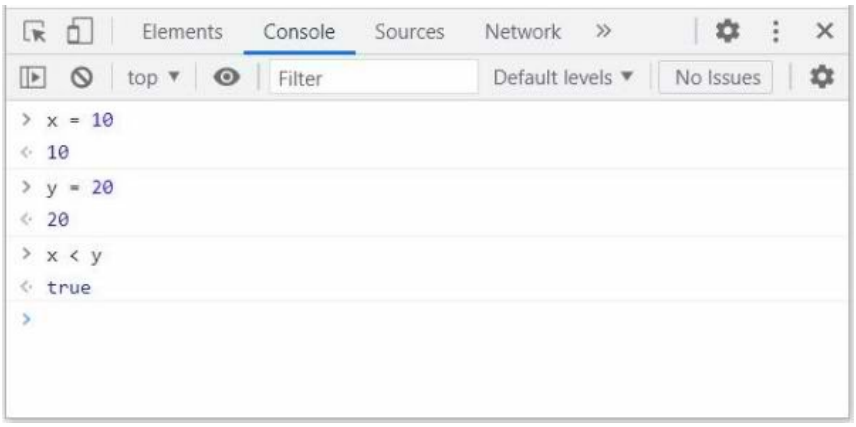
We can undock it into a separate window or only change the dock side following these steps:

- Click the "**Customize and control Dev Tools**" button in the upper-right-hand corner.
- Then select the **dock side** you want, as shown in the picture below.



2.2 Using Console Tool

You might have to select the **Console** tab if the Console tool wasn't selected. We can check the values of variables or try JavaScript code directly in the Console tool as below.

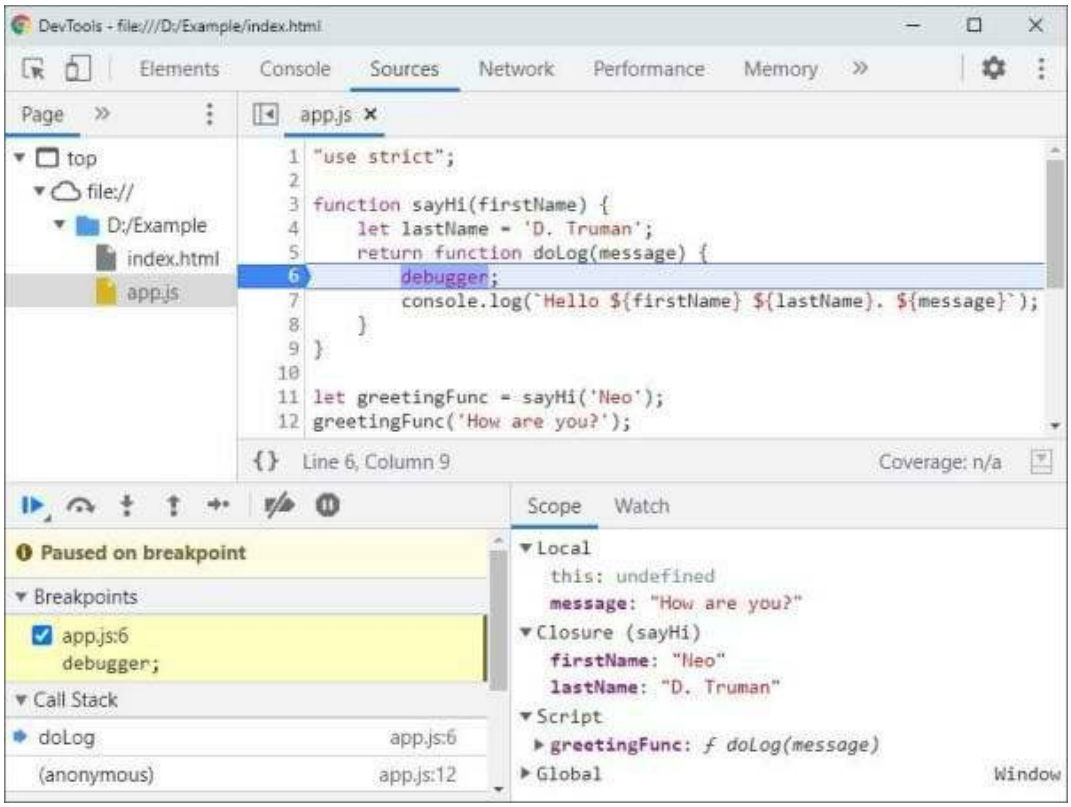


You can see the errors on your web page and the debugging messages here. The Console panel is the one that shows the messages you output with console.log() and any unhandled errors.

2.3 Using Source Tool

Select the **Sources** tab to open the debugging tool. It supports line-by-line debugging with breakpoints, reviewing the call stack, etc. The details will be

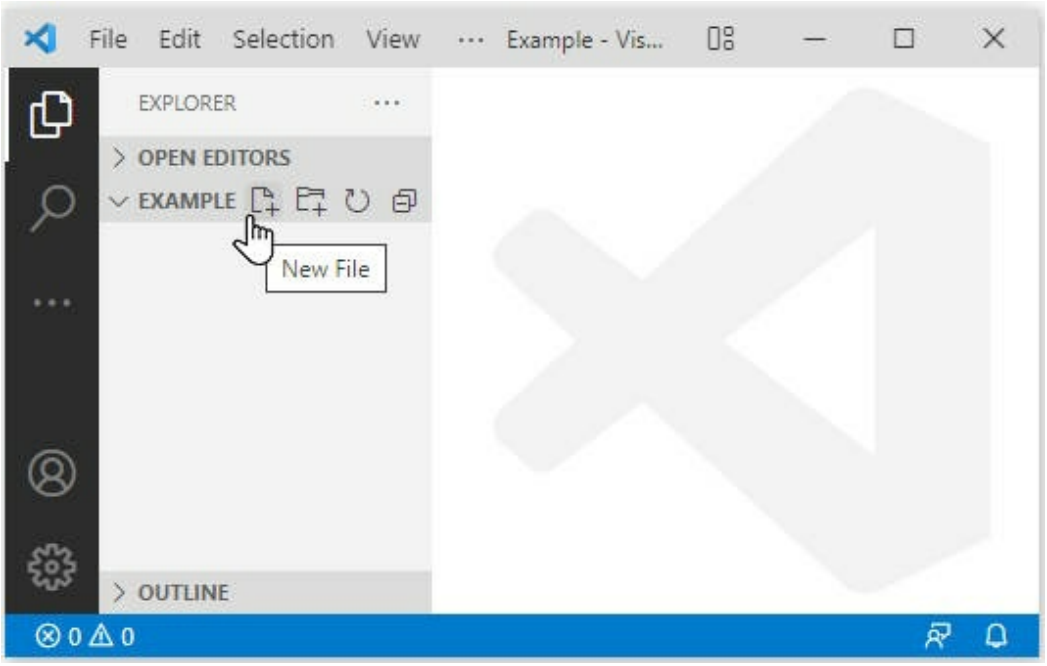
introduced with the code later.



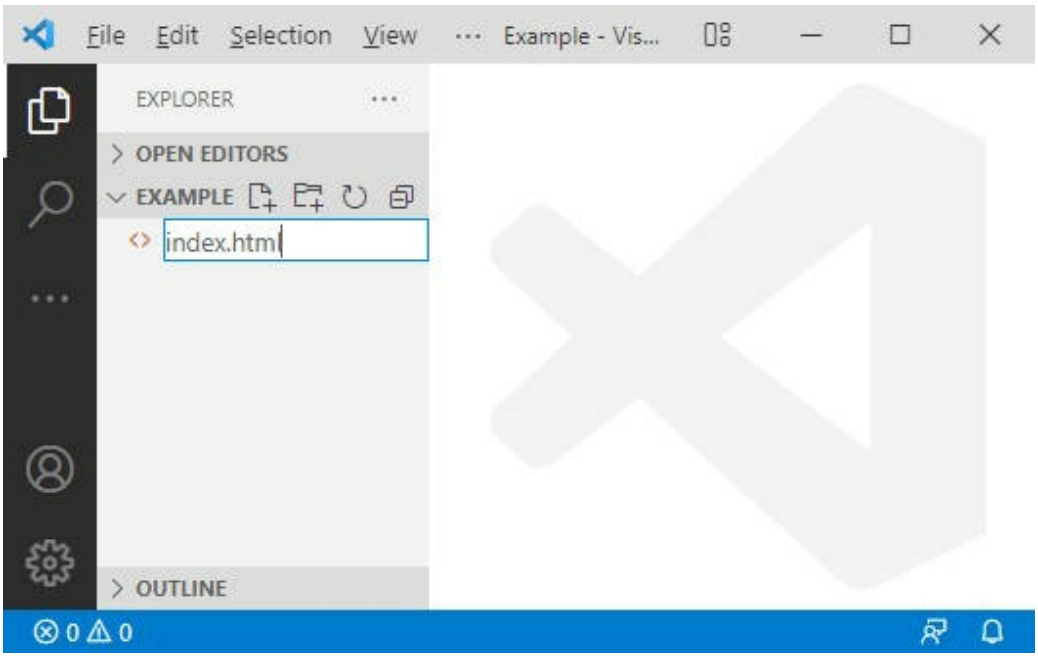
Preparing a Workspace

Please follow these steps to create your workspace:

- Create the *Example* folder, such as "*D:\Example*"
- Open the Visual Studio Code
- In the VS Code, select the menu **File > Open Folder…**
- Select your folder, "*D:\Example*" in my example
- Hover the mouse on the **EXAMPLE** folder; some icons will appear, as shown in the screenshot below.



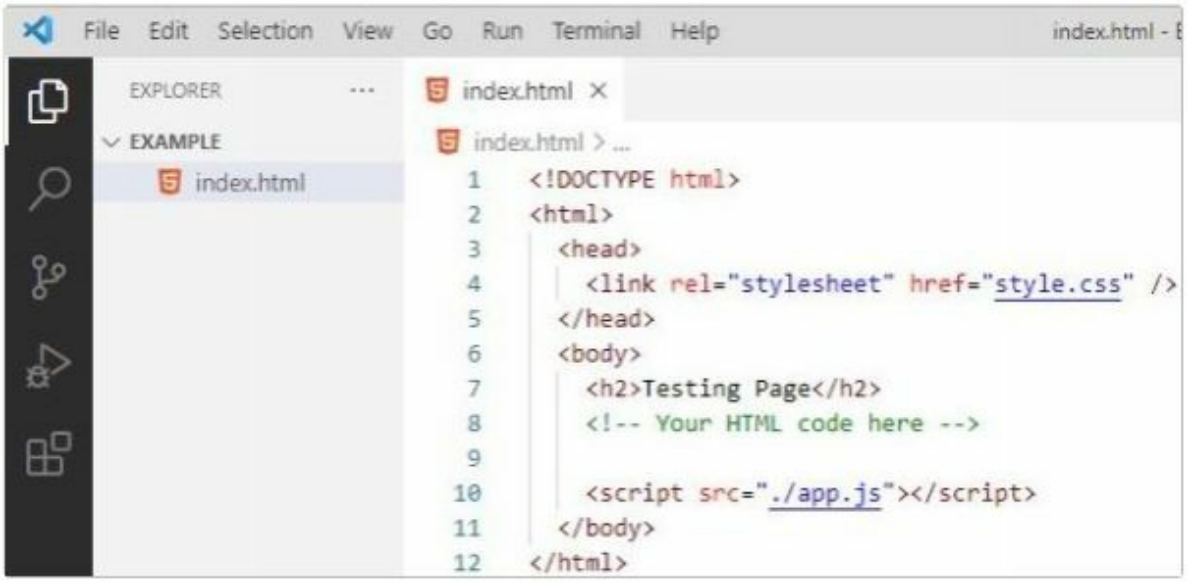
- Click the **New File** icon.



- Type "*index.html*" and press Enter
- Then type below HTML code into the editor

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h2>Testing Page</h2>
    <!-- Your HTML code here -->
  </body>
</html>
```

- Select menu **File > Save** to save the change to the *index.html* file.

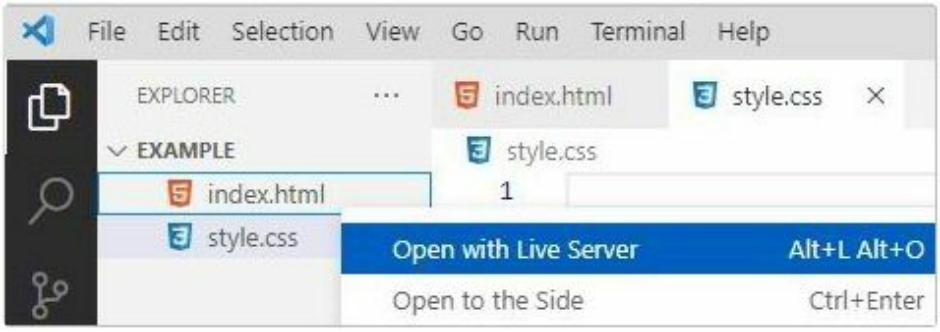


- Hover the mouse on the **EXAMPLE** folder and click the **New File** icon.
- Type "*style.css*" and press Enter but leave the file empty.
- Select menu **File > Save** to save the change to the *style.css* file.

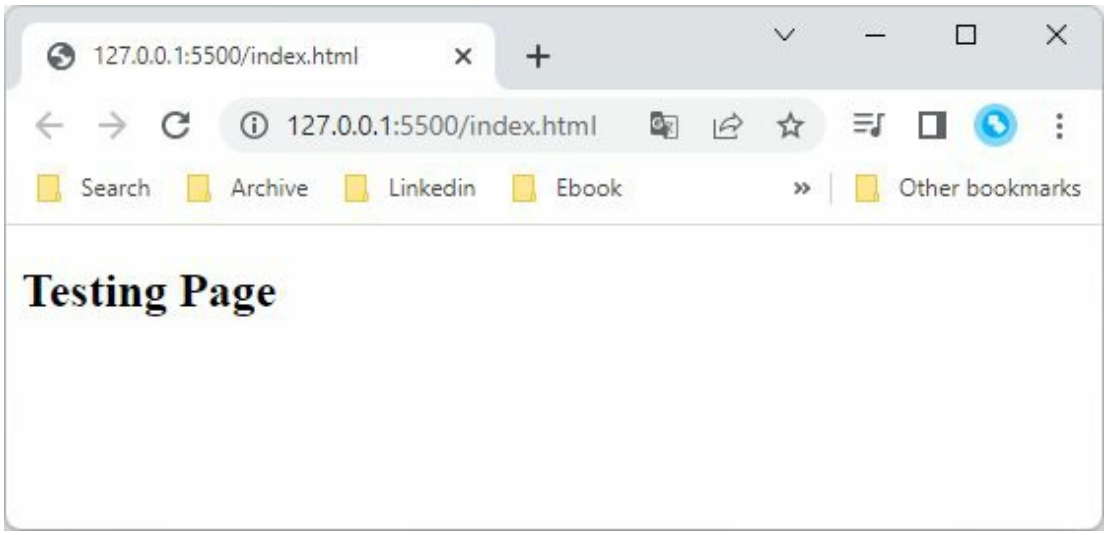
While learning this book, you will change the HTML code in the *index.html* file and the CSS code in the *style.css* file.

Let's try the code in a web browser like Google Chrome. Before following the steps, you should configure Google Chrome as your default web browser.

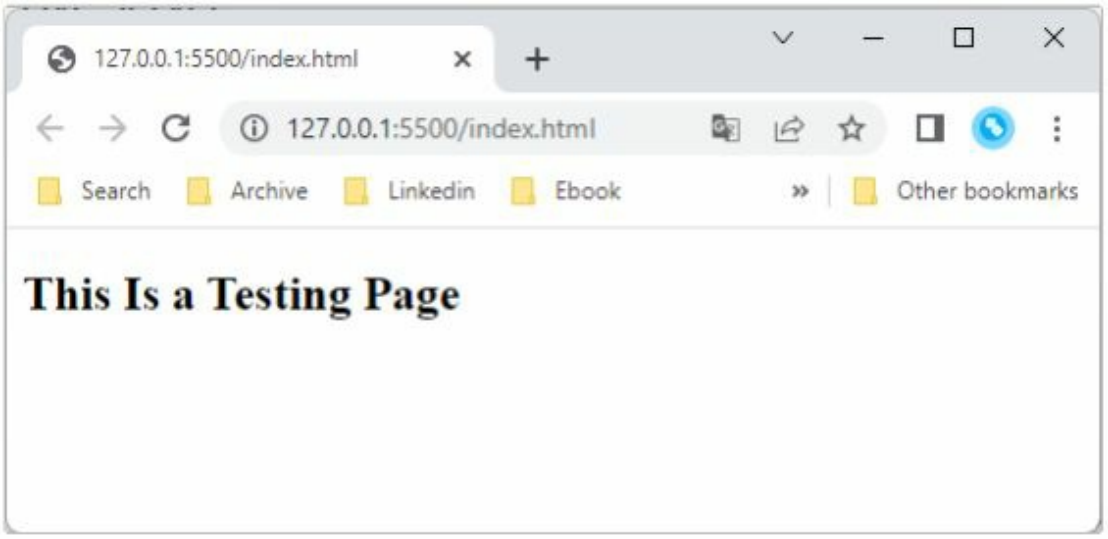
Right-click the *index.html* file from **Explorer Window** and click on **Open with Live Server**.



It will launch a local web server with a live reload feature for the file *index.html*.



We could change our HTML code and save the *index.html* file. Consequently, the web page will load the change for us, as shown in this screenshot.



Part I: HTML5

HTML is HyperText Markup Language for documents that are displayed in web browsers. There could be CSS and JavaScript code in the HTML documents to make the documents more attractive and support user interaction.

An HTML document includes many semantic elements to describe the structure of a webpage, such as headings, paragraphs, etc. Web browsers request and receive it from a web server, then render it into a webpage.

What will you learn?

In this part of the book, you will learn the following:

- HTML document structure and HTML elements such as headings, paragraphs, hyperlinks, images, lists, tables, iframes, etc.
- HTML entities, Data URIs
- HTML formatting elements and the style attribute
- Semantic HTML elements
- Block elements, inline elements, and inline-block elements
- Web forms, including form elements and form validation
- HTML multimedia: audio and video
- HTML graphics: SVG and canvas
- HTML events: window events, form events, keyboard events, mouse events, and clipboard events
- HTML APIs: drag and drop, geolocation, web storage, web workers.

CHAPTER 1
Getting to Know HTML

HTML stands for **H**yper**T**ext **M**arkup **L**anguage. It helps us describe the content and structure of our web pages. We use different HTML elements (or HTML tags) to describe different types of content, such as headings, paragraphs, links, images, etc.

Two Types of HTML Elements

An HTML element typically has three portions, for example:

```
<h1>My First Heading</h1>
```

- Opening tag: <h1>
- Content: “My First Heading”
- Closing tag: </h1> . The closing tag is the same as the opening tag but with a slash (/)

The closing tags will be omitted when elements do not have content, for example:

```
<img src="an-apple.jpg" alt="An apple" />
```

HTML Document Structure

The document should begin with a DOCTYPE element to tell the browser that this document uses HTML.

```
<!DOCTYPE html>
```

Then, we create an <html> element, which is the root element of the HTML document. It has two children, as below.

```
<html>
  <head></head>
  <body></body>
</html>
```

The above structure is always the same in all web pages, where the <head> element is for things that are not visible in the browser as it will contain the page title, some metadata of the page, links to CSS files, etc.

On the other hand, all the visible elements are placed inside the <body> element so that they will appear on the web pages. We often link our JavaScript files at the end of the <body> tag.

Your First Webpage

Let's add two more elements. The first element is the <title> element, which is used to set the webpage's title and is shown in the browser's title tab. The second one is the <h1> element, which is the main heading for the webpage, so we should only use it once.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Welcome to my site!</h1>
  </body>
</html>
```

Now, open the *index.html* file in a browser by double-clicking it or opening it with the Live Server. You will see the "**Welcome to my site!**" heading on the page.

Character Encoding

We use the “ charset ” attribute of the <meta> element, which will be placed inside the <head> element, to specify the character encoding for our HTML documents.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>My Website</title>
  </head>
  <body>
    <h1>Welcome to my site!</h1>
  </body>
</html>
```

The UTF-8 character set covers almost all of the characters and symbols in the world. Naturally, therefore, we should use it on our web pages.

Element Attributes

Various HTML elements have different attributes to configure the elements, set their contents, or adjust their behavior.

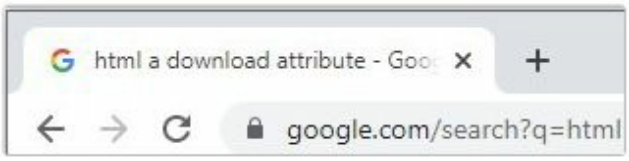
For example, we use the “ lang ” attribute to specify the language of our web pages as below.

```
<html lang="en">
```

In the above example, “ en ” stands for “English.”

Favorite Icon

A favorite icon is a small image displayed to the left of the page title in the browser tab, like this:



Do these steps to add a favorite icon to your website:

- prepare an ICO file as your favorite icon (A common name for it is “favicon.ico,” but not mandatory)
- save the file in the root directory or its sub-folder
- finally, add a <link> element to your “*index.html*” file, like the below example.

Example

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="icon" type="image/x-icon" href="/images/favicon.ico" />
 </head>
 <body>
  <h1>Welcome to my site!</h1>
 </body>
</html>
```

If the favorite icon were placed inside the root folder, the <link> element would change its href attribute as below.

```
<link rel="icon" type="image/x-icon" href="/favicon.ico" />
```

Comments

Comments are used to explain code and to make it more readable. They can also be used to prevent execution when testing alternative code.

In HTML, we use “ <!-- ” and “ --> ” to comment out code by wrapping the code by them as in the below example.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Heading 1</h1>
    <!-- <h2>Heading 2</h2>
    <h3>Heading 3</h3> -->
  </body>
</html>
```

We could only see the <h1> heading on the webpage.

CHAPTER 2
HTML Elements

. Headings

There are six headings from <h1> to <h6>, as “h” stands for “heading.” The <h1> tag is the most important heading and has the biggest font size. In contrast, <h6> defines the least important heading and is the smallest one.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <h1>Heading 1</h1>
  <h2>Heading 2</h2>
  <h3>Heading 3</h3>
  <h4>Heading 4</h4>
  <h5>Heading 5</h5>
  <h6>Heading 6</h6>
 </body>
</html>
```

It can be seen on the “*index.html*” page that there are six headings in different sizes.

! Paragraphs

We use <p> elements to create paragraphs since “p” stands for “paragraph.”

index.html

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Heading 1</h1>
    <p>This is your first paragraph.</p>
  </body>
</html>
```

3 Hyperlinks

The hyperlink is one of the fundamental HTML elements. A hyperlink (or “link” for short) contains an URL attribute so that links can help users navigate through a website. Without links, visitors do not know how to go to other web pages since they do not know their addresses.

To create a link, we use the <a> element which stands for “anchor.” The content between the opening and closing tags is the text displayed on the webpage.

3.1 External Anchors

The external anchors will navigate users to another webpage. This page can belong to the same website, or it can be to another website. The URL that this link points to is set using the “ href ” attribute as below.

Without the “ href ” attribute, an anchor element is not a link.

/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Home Page</h1>
    <a href="/contact.html">Contact Page</a>
    <a href="/user/login.html">Login Page</a>
  </body>
</html>
```

/contact.html

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

```
    <title>Contact Page</title>
  </head>
  <body>
    <h1>This is Contact Page</h1>
    <a href="/">Home Page</a>
  </body>
</html>
```

/user/login.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Login Page</title>
  </head>
  <body>
    <h1>This is Login Page</h1>
    <a href="/">Home Page</a>
  </body>
</html>
```

Now, users can navigate between the “Home” page, the “Contact” page, and the “Login” page using those links. Since *index.html* is the main page, we can write “ / ” instead of “ /index.html .”

To open an URL in a new tab, we can specify the “ target ” attribute as “ _blank .”

/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Home Page</h1>
    <a href="/contact.html" target="_blank">Contact Page</a>
    <a href="/user/login.html">Login Page</a>
  </body>
</html>
```

Moreover, we can set the “ href ” attribute to an URL of another website, such as “*https://www.google.com*.”

3.2 Internal Anchors

Internal anchors help users navigate within a webpage based on the IDs of its HTML elements. Without ID, the internal anchors will navigate users to the top of the current page. The syntax is:

```
<a href="#an-id-here">Description Here</a>
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Home Page</h1>
    <a href="#cake">Go to My Cake</a>

    <h2 id="apple">My Apple</h2>
    <img src="an-apple.jpg" alt="This is an apple" height="1000" />

    <h2 id="cake">My Cake</h2>
    <img src="a-cake.jpg" alt="This is a cake" height="1000" />

    <a href="#apple">Go to My Apple</a>
    <a href="#">Back to the top</a>
  </body>
</html>
```

Please prepare two images, “*an-apple.jpg*” and “*a-cake.jpg*,” by yourself and put them in the same folder containing the *index.html* file.

At the top of the page, clicking the anchor “**Go to My Cake**” will bring the heading “**My Cake**” to the viewport. Besides, clicking the anchor “**Back to the top**” will bring us to the top of the page.

3.3 Download Links

The “ download ” attribute of the <a> tag specifies that the target file (specified in the “ href ” attribute) will be downloaded when users click on the hyperlink.

We can declare a new name for the file by setting the value of the “ download ” attribute, but this is optional. Moreover, if we do not specify the file extension, the browser automatically detects and adds it to the file.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <a href="/my-img.jpg" download="new-name">Download Image</a>
  </body>
</html>
```

Images

To display images, we use the <img> elements. <img> has no closing tag since it has no content. Therefore, we add the slash at the very end of the element like this:

```
<img />
```

To specify which image to be displayed, we use the " src " (which stands for "source") attribute to describe the path of the image file.

```
<img src="an-apple.jpg" />
```

Another important attribute is " alt ," which stands for "alternative text" and describes what the image is about. This attribute is essential for the following:

- search engines such as Google actually to know what content the image is
- and allow the screen reader to read it out loud

```
<img src="an-apple.jpg" alt="This is an apple" />
```

- the alternative text will be used when the " src " attribute is invalid. Because the path is invalid, the browser cannot download the image, so that it will display the image description from the " alt " attribute.

```
<img src="invalid-image.pg" alt="This is an apple" />
```

Two more attributes are usually used in the <img> element; they are " width " and " height ." However, we only use one to scale the image up or down. Otherwise, using both will distort the image because our width and height may not be in the same ratio as the original image size.

Let's examine a square image of 300x300 pixels.

- This code will scale down the image:

```
<img src="my-image.jpg" alt="This is an apple" width="100" />
```

or

```
<img src="my-image.jpg" alt="This is an apple" height="100" />
```

- This code will scale up the image:

```
<img src="my-image.jpg" alt="This is an apple" width="500" />
```

or

```
<img src="my-image.jpg" alt="This is an apple" height="500" />
```

- And this will distort the image:

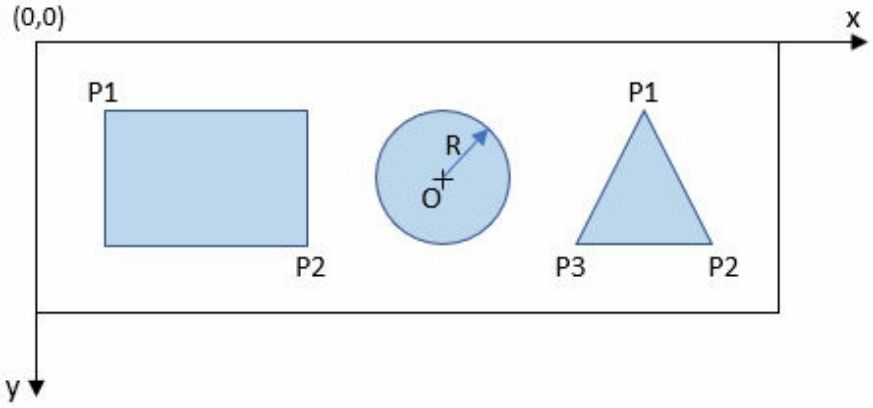
```
<img src="my-image.jpg" alt="This is an apple" width="200" height="100" />
```

4.1 Image Maps

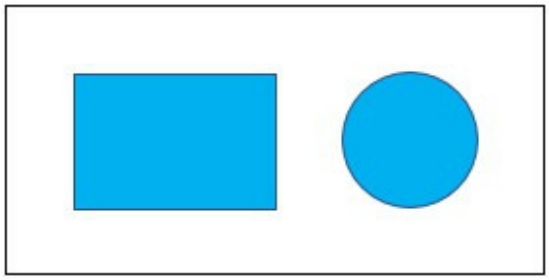
The <map> element defines an image map having clickable areas. The areas are represented with one or more <area> elements as children of the <map> element. Every <map> has a " name " attribute, so the <img> element can use its " usemap " attribute to specify which map to use.

An <area> element uses " shape ," and " cords " attributes to define a clickable area inside the image. The " shape " attribute can have one of these values:

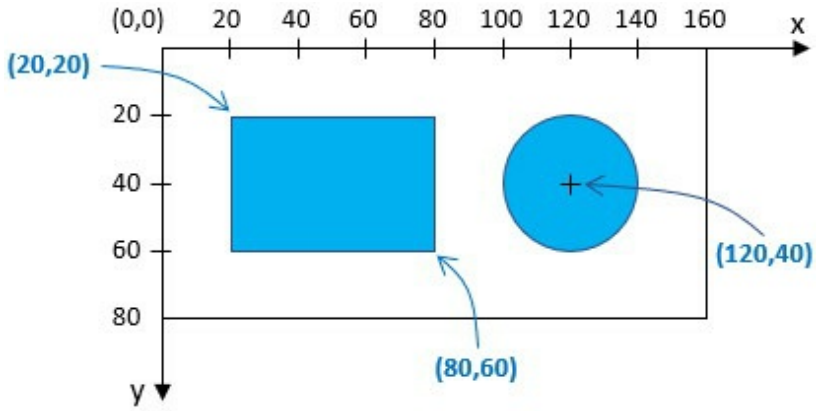
- circle - a circle. The " cords " attribute will be "O_x, O_y, R."
- rect - a rectangle. The " cords " attribute will be "$P1_x$, $P1_y$, $P2_x$, $P2_y$."
- poly - a polygon. The " cords " attribute will be "$P1_x$, $P1_y$, $P2_x$, $P2_{y, \ldots,}$ PN_x, PN_y."



For example, we will use this image for testing the image map:



and the image's coordinates are as below.



index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <img src="navbar.jpg" alt="Navigation bar" usemap="#nav-bar" width="160" height="80" />

    <map name="nav-bar">
      <area shape="rect" coords="20,20,80,60" alt="Computer" href="#cake" />
      <area shape="circle" coords="120,40,20" alt="Coffee" href="#apple" />
    </map>

    <h2 id="apple">My Apple</h2>
    <img src="an-apple.jpg" alt="This is an apple" height="1000" />

    <h2 id="cake">My Cake</h2>
    <img src="a-cake.jpg" alt="This is a cake" height="1000" />
  </body>
</html>
```

4.2 Responsive Pictures

The <picture> element contains multiple images, and only one of them will be chosen to display based on the media queries of the <source> tags. If none of the source tags matches, the <img> element will be used as a fallback option.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>Resize the browser to load different images.</p>
  <picture>
   <source media="(min-width:500px)" srcset="3.jpg" />
   <source media="(min-width:300px)" srcset="2.jpg" />
   <img src="1.jpg" alt="A number" />
  </picture>
 </body>
</html>
```

5 Lists

5.1 Unordered List

We make a list of bullet points using the <ul> element, which stands for "unordered list." The list items are made by using the <li> elements which stand for "list item," and they are nested inside the <ul> element as below.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ul>
  </body>
</html>
```

On the webpage, we will see the following:

- Item 1
- Item 2
- Item 3

5.2 Ordered List

We make an ordered list with numbers using the <ol> element, which stands for “ordered list.” The list items are made by using the <li> elements which stand for “list item,” and they are put inside of the <ol> element as below.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <ol>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ol>
  </body>
</html>
```

On the webpage, we will see the following:

1. Item 1
2. Item 2
3. Item 3

5.3 Description Lists

We use one <dl> element to make a description list. A description in the list is constructed by using these two elements:

- <dt> - defines terms/names
- <dd> - describes each term/name

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <dl>
   <dt>HTML</dt>
   <dd>is the standard markup language for Web pages.</dd>
   <dt>CSS</dt>
   <dd>is the language we use to style an HTML document.</dd>
   <dt>JavaScript</dt>
   <dd>is the world's most popular programming language.</dd>
  </dl>
 </body>
</html>
```

Result:

HTML
  is the standard markup language for Web pages.
CSS
  is the language we use to style an HTML document.
JavaScript
  is the world's most popular programming language.

5 Tables

When arranging data into rows and columns, we use the <table> elements. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <style>
      table {
        border-collapse: collapse;
      }
      th,
      td {
        border: 1px solid black;
      }
      .days {
        background-color: lightyellow;
      }
      .weekend {
        background-color: lightblue;
      }
    </style>
  </head>
  <body>
    <table>
      <caption>
        September 2022
      </caption>
      <colgroup>
        <col span="5" class="days" />
        <col span="2" class="weekend" />
      </colgroup>
      <thead>
        <tr>
          <th>Mon</th>
          <th>Tue</th>
          <th>Wed</th>
          <th>Thu</th>
          <th>Fri</th>
          <th>Sat</th>
```

```
    <th>Sun</th>
  </tr>
 </thead>
 <tbody>
  <tr>
    <td></td>
    <td></td>
    <td></td>
    <td>1</td>
    <td>2</td>
    <td>3</td>
    <td>4</td>
  </tr>
  <tr>
    <td>5</td>
    <td>6</td>
    <td>7</td>
    <td>8</td>
    <td>9</td>
    <td>10</td>
    <td>11</td>
  </tr>
  <tr>
    <td>12</td>
    <td>13</td>
    <td>14</td>
    <td>15</td>
    <td>16</td>
    <td>17</td>
    <td>18</td>
  </tr>
  <tr>
    <td>19</td>
    <td>20</td>
    <td>21</td>
    <td>22</td>
    <td>23</td>
    <td>24</td>
    <td>25</td>
  </tr>
  <tr>
    <td>26</td>
    <td>27</td>
    <td>28</td>
    <td>29</td>
```

```
        <td>30</td>
        <td colspan="2"></td>
       </tr>
     </tbody>
   </table>
  </body>
</html>
```

Result:

September 2022

Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

6.1 Table Headers

In the above example, a table header is a <tr> element that contains some <th> elements. Moreover, we often put this <tr> element inside a <thead> element. Table headers are optional.

6.2 Table Rows

A table row is a <tr> element that contains some <td> elements, and we often put this <tr> element inside a <tbody> element, as in the above example.

6.3 Table Caption

We use the <caption> element to set the table's caption. It is usually inserted immediately after the <table> tag. By default, this caption will be center-aligned above the table.

6.4 Group of Columns

The <colgroup> element selects groups of columns in a table for formatting. It contains one or more <col> elements that specify the groups (one <col> is one group). We often use the " span " attribute of the <col> element to group some columns and the " class " attribute to format the table cells by class names in the CSS code.

6.5 Column Span

The " colspan " attribute of the <td> element specifies the number of columns a cell should span in the table. Besides, <td> also has the " rowspan " attribute for merging the cells vertically.

7 Breaks

7.1 Line Breaking

The
 element inserts a single line break. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>I've added a line break here.<br />This sentence will be on the second line.</p>
  </body>
</html>
```

7.2 Thematic Breaking

The <hr> element defines a thematic break in a webpage, e.g., a topic shift. It is displayed as a horizontal rule that is used to separate content. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Web Development</h1>
    <p>HTML is the standard markup language for web pages.</p>
    <hr />
    <p>CSS is the language we use to style an HTML document.</p>
    <hr />
    <p>JavaScript is the world's most popular programming language.</p>
  </body>
</html>
```

8 Progress Bars

8.1 <progress>

The <progress> element help us show the completion progress of a task. The " value " attribute is the current progress, while the " max " attribute is often set to 100 . For example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>
   Uploading:
   <progress value="60" max="100"></progress>
   60%
  </p>
 </body>
</html>
```

Result:

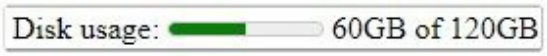
8.2 <meter>

The <meter> element defines a scalar measurement within a known range, such as dish usage. The “ value ” attribute is the current meter’s value, while the “ max ” attribute is the maximum value of the meter. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>
      Disk usage:
      <meter value="60" max="120"></meter>
      60GB of 120GB
    </p>
  </body>
</html>
```

Result:



9 Computer Code

9.1 <code>

The <code> element defines computer code. It will be displayed in the default monospace font of the browser. For example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>This is a piece of JavaScript code:</p>
  <code>var x = 10;</code>
 </body>
</html>
```

9.2 <pre>

Text in a <pre> element is displayed in the browser's default monospace font. Moreover, the text will be displayed as written in the HTML code. It means that both spaces and line breaks will be kept unchanged. For example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <pre>
   Text in a pre element is displayed
   in the browser's default monospace font,
   and it preserves both      spaces
   and line breaks
  </pre>
 </body>
</html>
```

.0 Iframes

An <iframe> element displays another webpage within the current HTML document. Its “ src ” attribute sets the URL to be loaded into the iframe. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <style>
      #my-iframe {
        width: 800px;
        height: 600px;
        border: none;
      }
    </style>
  </head>
  <body>
    <h1>Web Development Books</h1>
    <iframe id="my-iframe" src="https://www.amazon.com/dp/B09VFLS7TF"></iframe>
  </body>
</html>
```

By default, an <iframe> element has a border around it. Therefore, we often use the CSS “ border ” property to remove it, as in the above example.

.1 HTML Entities

In HTML, there are some reserved characters. For example, we cannot use the less than (<) or greater than (>) signs in our text since the browsers might mix them with HTML tags.

To display reserved characters in HTML, we use HTML entities. The syntax of an HTML entity looks like this:

```
&entity-name;
```

or

```
&#entity-number;
```

These are some popular HTML entities:

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
¢	cent	¢	¢
£	pound	£	£
¥	yen	¥	¥
€	euro	€	€
©	copyright	©	©
®	registered trademark	®	®
™	trademark	™	™
←	leftwards arrow	←	←
↑	upwards arrow	↑	↑
→	rightwards arrow	→	→

↓	downwards arrow	↓	↓

Below is an example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <footer>&copy; 2022, Neo D. Truman</footer>
  </body>
</html>
```

CHAPTER 3
HTML Styles

Formatting Elements

1.1 <b>

We can make some text in bold by putting it inside a <b> element as “b” stands for “bold.” For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>This is your <b>first paragraph</b></p>
  </body>
</html>
```

The text “first paragraph” will turn bold when we reload the webpage.

1.2 <i>

Besides, we can wrap some text that we want to be italic inside of the <i> element as “i” stands for “italic.” For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
```

```
    <p>This is your <i>first paragraph</i></p>
  </body>
</html>
```

1.3 <u> or <ins>

The <u> or <ins> element represents some text displayed with an underline. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>This is some <u>mispeled</u> text.</p>
  </body>
</html>
```

1.4 <s> or <del>

The <s> or <del> element shows text that is not accurate and will be displayed with a line through it. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p><s>These are</s>This is some text.</p>
  </body>
</html>
```

1.5 <mark>

The <mark> element defines the text that should be highlighted by changing its background color. For example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>This is an <mark>important</mark> paragraph</p>
 </body>
</html>
```

1.6 <sub> and <sup>

The <sub> element defines subscript text which appears half a character below the regular line and is rendered in a smaller font. For example, “1” in “X_1.”

The <sup> element defines superscript text which appears half a character above the normal line and is rendered in a smaller font. For example, “2” in “X^2.”

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>This is a <sub>subscript</sub> text.</p>
  <p>This is a <sup>superscript</sup> text.</p>
 </body>
</html>
```

HTML Style Attribute

CSS declaration code can be placed in the “ style ” attribute of HTML elements. These inline styles will be applied only to the element that declares the “ style ” attribute. In the below example, inline styles are applied only to the first paragraph.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p style="color: red; font-size: 26px">This is the first paragraph.</p>
    <p>This is the second paragraph.</p>
  </body>
</html>
```

However, inline CSS should be avoided due to these reasons:

- it pollutes the HTML code
- and it is hard to maintain

CHAPTER 4
Semantic HTML

Semantic HTML is a web document using certain HTML elements which have a meaning or a purpose attached to them. These elements do not render anything special in a browser since they have special meanings.

. Semantic Elements

1.1 <blockquote>

The <blockquote> element specifies a quoted portion from another website or source. URL of the source is defined using the “ cite ” attribute. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>Here is a quote from Wikipedia:</p>
    <blockquote cite="https://en.wikipedia.org/wiki/HTML">The HyperText Markup Language or
HTML is the standard markup language for documents designed to be displayed in a web browser.
</blockquote>
  </body>
</html>
```

Besides, the <q> tag defines a short quotation, although it is not a semantic element.

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

```
    <title>My Website</title>
  </head>
  <body>
    <p>Wikipedia's goal is to: <q>Provide a free encyclopedia that anyone can edit.</q></p>
  </body>
</html>
```

1.2 <address>

To define the contact information of the author of a document or an article, we use the <address> element. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <address>
      <p>12345 South St. Philadelphia, PA 12345</p>
      <p>
        <a href="tel:123-456-7890">123-456-7890</a><br />
        <a href="mailto:contact@neodtruman.com">contact@neodtruman.com</a>
      </p>
    </address>
  </body>
</html>
```

1.3 <details> and <summary>

The <details> element is used to create a widget that users can open and close (it is closed by default). When opened, it shows the content within.

The <summary> element is a child of <details> and defines the heading for the details. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <details>
      <summary>HTML</summary>
      <p>HTML is the standard markup language for Web pages.</p>
    </details>
  </body>
</html>
```

1.4 <figure> and <figcaption>

A <figure> element defines a photo in the HTML documents. Besides, we often use the <figcaption> element to add a caption for it. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <figure>
      <img src="an-apple.jpg" alt="An apple" />
      <figcaption>Fig.1 - My apple</figcaption>
    </figure>
  </body>
</html>
```

Semantic Layout Elements

HTML has several semantic elements that define different portions of a webpage. They are:

- <article> element defines independent content.
- <main> element specifies the main content of the HTML document.
- <section> element defines a section in an HTML document.
- <header> element is the top part of a web document or the top part of some more minor elements like <section> and <article> .
- <footer> element is used to wrap content that comes at the very end of a page, such as the copyright or at the end of <section> and <article> .
- <aside> element represents a section of a page that consists of content that is related to the main content.
- <nav> stands for “navigation,” and it is where we put our navigation links.

Example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <header>
   <nav>
    <a href="#">Logo</a>
    <ul>
     <li><a href="#">Posts</a></li>
     <li><a href="#">Contact</a></li>
     <li><a href="#">Login</a></li>
    </ul>
   </nav>
   <h1>Main Heading</h1>
  </header>
```

```
  <main>
    <section>
      <h2>Section's Heading 1</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut eget ex eget lacus luctus porta ac ut nisi. Nunc euismod maximus faucibus. In hac habitasse platea dictumst. Sed molestie consequat massa, non sollicitudin metus dapibus eu.</p>
    </section>

    <section>
      <h2>Section's Heading 2</h2>
      <p>In egestas dictum suscipit. Vestibulum condimentum tristique imperdiet. Quisque lacinia eros et tempus volutpat. Morbi malesuada eleifend sapien, id tristique urna porta maximus.</p>
    </section>
  </main>

  <footer>
    <p>Copyright &copy; 2022 by Neo D. Truman</p>
    <address>
      <p>12345 South St. Philadelphia, PA 12345</p>
      <p>
        <a href="tel:123-456-7890">123-456-7890</a><br />
        <a href="mailto:contact@neodtruman.com">contact@neodtruman.com</a>
      </p>
    </address>
  </footer>
 </body>
</html>
```

3 Semantic Formatting Elements

3.1

Instead of using the <b> tag, we should use the element because it has semantic meaning. A element means a critical piece of content that wants to stand out from the page. For example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
```

```
  </head>
  <body>
    <h1>Heading 1</h1>
    <p>This is your <strong>first paragraph</strong></p>
  </body>
</html>
```

3.2

The element emphasizes a piece of content since it stands for “emphasize.” For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Heading 1</h1>
    <p>This is your <em>first paragraph</em></p>
  </body>
</html>
```

CHAPTER 5
Web Forms

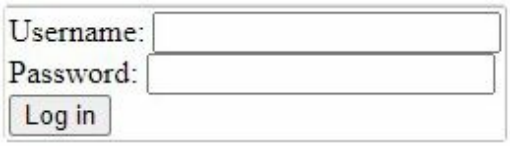
We use HTML forms to collect user input and send it to our server for processing. A web form might contain many HTML input elements to collect the user input. Each input usually has a label describing what information the user should enter.

Below is a simple form that we usually see on web pages:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <form action="/login">
   <label for="acc-name">Username:</label>
   <input type="text" id="acc-name" name="username" /><br />
   <label for="acc-pass">Password:</label>
   <input type="password" id="acc-pass" name="password" /><br />
   <input type="submit" value="Log in" />
  </form>
 </body>
</html>
```

Result:



Enter some text into the boxes and click the "**Log in**" button, and you will be redirected to the "*/login*" page. For example, I entered " john " as my username and " 123 " as my password, so I have been redirected to this URL:

http://127.0.0.1:5500/login?username=john&password=123

It also means that the browser has sent an HTTP request to the web server to retrieve the webpage at the address "*http://127.0.0.1:5500/login*," including a query string, which in this case is " username=john&password=123 ." In the query string, " username " and " password " are specified by the " name " attributes of the <input> elements. Finally, the server will authenticate users using this username and password.

HTML Form Elements

1.1 <input> and <label>

The <input> element is used to collect user input. There are many types of inputs, and they will be introduced in the next section.

Besides, a <label> element is often used to describe what information the user should enter into the <input> tag. The " for " attribute of <label> specifies which form element this label is bound to by using the element's ID. As a result, the input will get focused when the user clicks on the label.

Example:

```
<label for="fname">First name:</label>
<input type="text" id="fname" name="firstname" />
```

Result:

1.2 <textarea>

The <textarea> element defines a multi-line text input control.

Example:

```
<textarea name="comment" rows="3" cols="30"> </textarea>
```

Result:

Users can enter as many lines of text as they want into the element. Moreover, they can drag and drop the bottom-right corner of the <textarea> element to resize it.

1.3 <select> and <option>

The <select> element is used to create a drop-down list containing many

options. Each option is defined by using an <option> element.

Example:

```
<select name="car">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="mercedes">Mercedes</option>
  <option value="audi">Audi</option>
</select>
```

Result:

We can use the <optgroup> element to group related options in a <select> element like this. The " label " attribute of <optgroup> will be rendered in the drop-down list to clarify the groups.

Example:

```
<select name="car">
  <optgroup label="Swedish Cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
  </optgroup>
  <optgroup label="German Cars">
    <option value="mercedes">Mercedes</option>
    <option value="audi">Audi</option>
  </optgroup>
</select>
```

Result:

1.4 <button>

The <button> element defines a clickable button. The button's text is placed between the opening and closing tags. When a user clicks a <button> element inside a form (<form>), all form values will be submitted to the server.

Example:

```
<button>Submit</button>
```

Result:

The " onclick " attribute can specify which JavaScript code will be executed when the user clicks the button, but it is optional. After executing the JS code, the form values will be submitted to the server. For example:

```
<button onclick="alert('Hi')">Submit</button>
```

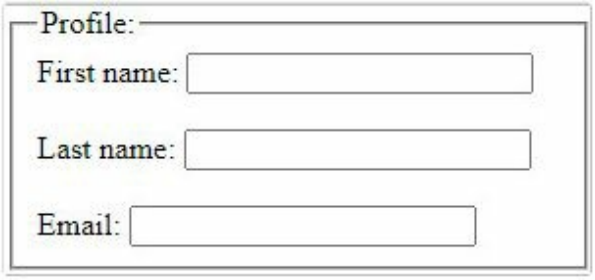
1.5 <fieldset> and <legend>

The <fieldset> element is used to group related elements in a form. It could have a <legend> tag inside to define a caption for the group of elements.

Example:

```
<fieldset>
  <legend>Profile:</legend>
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname" />
  <br /><br />
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname" />
  <br /><br />
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" />
</fieldset>
```

Result:



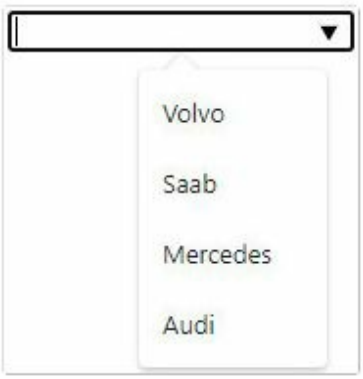
1.6 <datalist>

The <datalist> element specifies a list of pre-defined options for an <input> element.

Example:

```
<datalist id="car-list">
  <option value="Volvo"></option>
  <option value="Saab"></option>
  <option value="Mercedes"></option>
  <option value="Audi"></option>
</datalist>
<input type="text" name="car" list="car-list" />
```

Result:



2 HTML Input Types

2.1 Buttons

2.1.1 button

The <input type="button"> element is a clickable button. Its “ value ” attribute is used to set the button’s text.

This input type needs the “ onclick ” attribute to specify which JavaScript code will be executed when the user clicks the button. Without the attribute, this input type will do nothing when the user clicks it.

Example:

```
<input type="button" value="Click me" onclick="alert('Hi')" />
```

Result:

2.1.2 submit

The <input type="submit"> element is a submit button that submits all form values to the server. Its “ value ” attribute is used to set the button’s text, but it is optional since its default value is “ Submit .”

Example:

```
<input type="submit" />
```

Result:

2.1.3 reset

The <input type="reset"> element is a reset button that resets all form values to their initial values. Its “ value ” attribute is used to set the button’s text, but it is optional since its default value is “ Reset .”

Example:

```
<input type="reset" />
```

Result:

Reset

2.1.4 image

The <input type="image"> element is an image used as a submit button. This special button also submits the coordinates where the user clicks on the button, for example:

*http://127.0.0.1:5500/login?firstname=John&**x=36&y=25***

Example: (please prepare the “ btn-submit.jpg ” image by yourself)

```
<input type="image" src="btn-submit.jpg" alt="Submit" width="50" height="50" />
```

2.2 Texts

2.2.1 text

The <input type="text"> defines a single-line text field.

Example:

```
<input type="text" name="firstname" />
```

When we type “John” into the input field, we will see this:

2.2.2 password

The <input type="password"> defines a password field.

Example:

```
<input type="password" name="password" />
```

When we type “ 12345678 ,” into the input field, we will see this:

The input’s value is still “ 12345678 ” but the browser hides it since this input is a password field.

2.2.3 email

The <input type="email"> defines a field for an email address.

Example:

```
<input type="email" name="email" />
```

This email input differs from the text input since it has a simple email validation. When a user clicks the submit button, the browser will validate the user input whether is a valid email or not. For example:

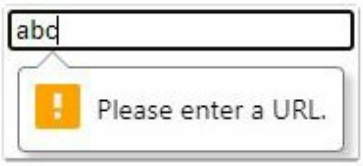
2.2.4 url

The <input type="url"> defines a field for entering a URL.

Example:

```
<input type="url" name="yourSite" />
```

This URL input differs from the text input since it has a simple URL validation. When a user clicks the submit button, the browser will validate user input whether it is a valid URL or not. For example:



2.2.5 hidden

The <input type="hidden"> defines a hidden input field. Although it is not displayed on the webpage, its value will be submitted to the web server when the user clicks the submit button.

Example:

```
<input type="hidden" name="userId" value="123" />
```

2.3 Numbers

2.3.1 number

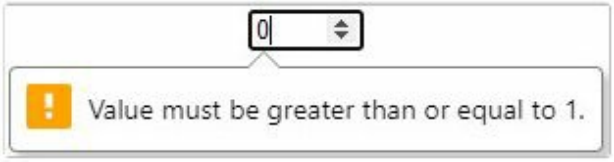
The <input type="number"> defines a field for entering a number, so the user can only enter numbers.

The “ min ” and “ max ” attributes are optional. We use them to tell the browser to validate user input before submitting it to the server.

Example:

```
<input type="number" name="quantity" min="1" max="9" />
```

Result:



2.3.2 range

The <input type="range"> defines a control for selecting a number via slider control.

Example:

```
<input type="range" name="points" min="0" max="10" />
```

Result:

2.4 Options

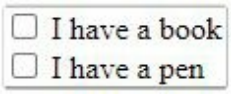
2.4.1 checkbox

The <input type="checkbox"> elements support selecting one or more options.

Example:

```
<input type="checkbox" id="book" name="book" value="Book" />
<label for="book"> I have a book</label>
<br />
<input type="checkbox" id="pen" name="pen" value="Pen" />
<label for="pen"> I have a pen</label>
```

Result:



We can select none, one, or both options in the above example.

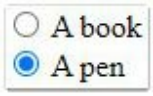
2.4.2 radio

The <input type="radio"> are radio buttons usually presented in several related options. Only one radio button in a group can be selected simultaneously.

So, in this example, we can select only one option:

```
<input type="radio" id="book" name="schoolThing" value="Book" />
<label for="book"> A book</label>
<br />
<input type="radio" id="pen" name="schoolThing" value="Pen" />
<label for="pen"> A pen</label>
```

Result:



2.5 Files

The <input type="file"> element defines a file-select field and a “Choose File” button for file uploads.

Example:

```
<input type="file" name="bgimage" />
```

Result:

2.6 Date and Time

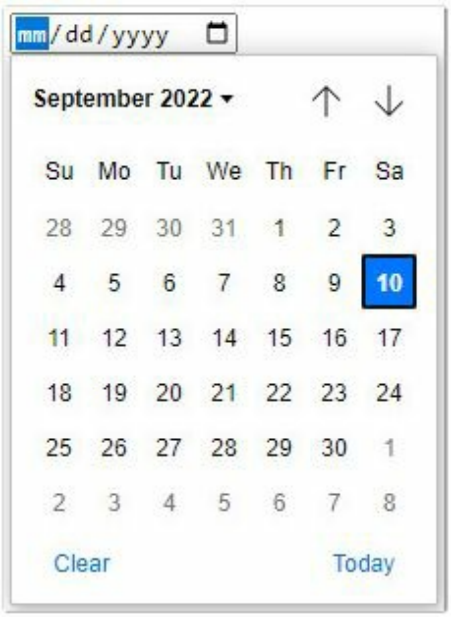
2.6.1 date

The <input type="date"> element defines a date picker.

Example:

```
<input type="date" name="birthday" />
```

Result:



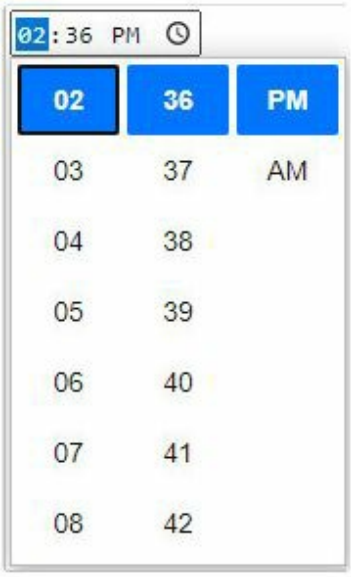
2.6.2 time

The <input type="time"> element defines a time picker.

Example:

```
<input type="time" name="alarmTime" />
```

Result:



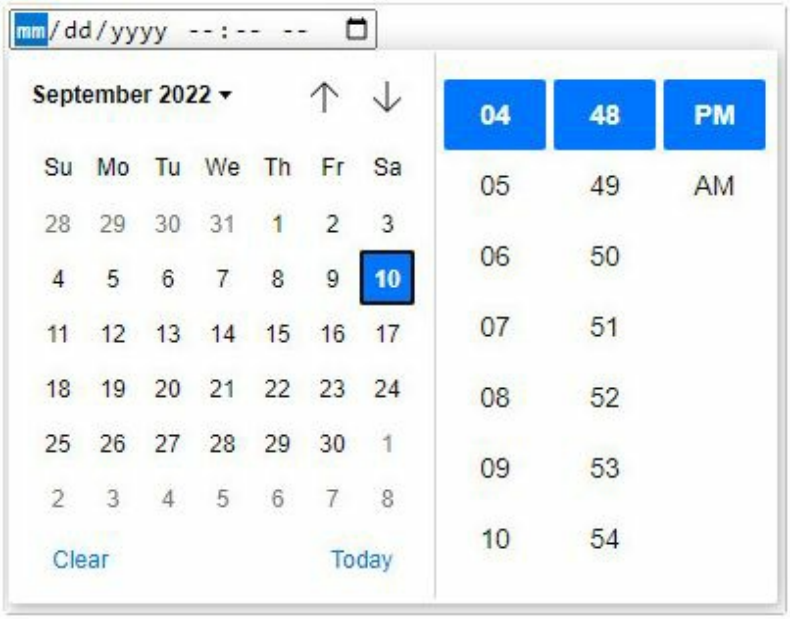
2.6.3 datetime-local

The <input type="datetime-local"> element defines a date-time picker.

Example:

```
<input type="datetime-local" name="alarmDateTime" />
```

Result:



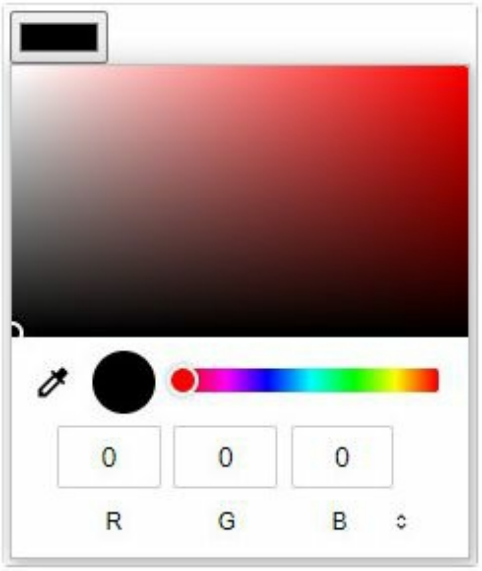
2.7 Colors

The <input type="color"> element defines a color picker.

Example:

```
<input type="color" name="bgcolor" />
```

Result:



3 HTML Input Attributes

3.1 name

The most important attribute of an HTML form element is “ name ” since it will be used as a parameter in the query string, which will be sent to the web server when the user submits the form. We have already known how it works at the beginning of this chapter.

3.2 value

The “ value ” attribute defines an initial value for an input element.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <form action="/login">
      <label for="fname">First name:</label>
      <input type="text" id="fname" name="firstname" value="John" />
    </form>
  </body>
</html>
```

Result:

3.3 placeholder

The “ placeholder ” attribute specifies a short hint describing an input element’s expected value. The hint will be replaced with user input as they type.

This attribute works with <input> (input types: text, password, email , and url) and <textarea> .

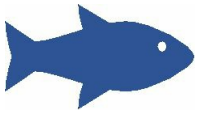
Example:

```
<input type="text" name="firstname" placeholder="Your first name here" />
```

Result:

3.4 readonly

The “ readonly ” attribute is used to make an element un-editable.



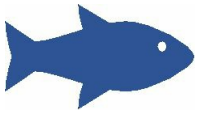
Although the element cannot be edited, its value will be submitted to the server when the user clicks the submit button.

Example:

```
<input type="text" name="firstname" value="John" readonly />
```

3.5 disabled

The “ disabled ” attribute makes an element unusable and un-clickable. This attribute can be applied to these elements: <input>, <button>, <textarea>, <select>, <option>, <optgroup> , and <fieldset> .



Values of disabled elements will not be submitted to the server when the user clicks the submit button.

Example:

```
<input type="text" name="firstname" value="John" disabled />

<input type="submit" disabled />
```

Result:

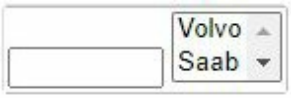
3.6 size

The “ size ” attribute specifies the width of an <input> element in characters. For <select> elements, the “ size ” attribute specifies the number of visible options in its drop-down list.

Example:

```
<input type="text" name="firstname" size="6" />

<select name="car" size="2">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="mercedes">Mercedes</option>
  <option value="audi">Audi</option>
</select>
```

Result:



3.7 multiple

The “ multiple ” attributes specify that users can enter more than one value in the <input> (input types: email and file) or <select> elements.

Example:

```
<select name="cars" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="mercedes">Mercedes</option>
  <option value="audi">Audi</option>
</select>
```

Result:

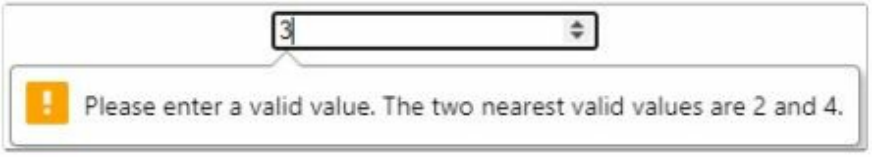
3.8 step

The “ step ” attribute specifies the interval between legal numbers in an <input> element. For example, if the step is 2, valid numbers could be -2, 0, 2, 4, 6, etc. This attribute is often applied to this input type: number .

Example:

```
<input type="number" name="points" step="2" />
```

Result:



3.9 width and height

The width and height attributes are used to set the size (in pixels) of these elements: <img>, <input type="image">, <video>, <canvas> , and <iframe> .

Example:

```
<iframe src="https://www.amazon.com/dp/B09VFLS7TF" width="800" height="600"></iframe>
```

3.10 autofocus

The autofocus attribute specifies that an element should get focused when the page loads. This attribute can be applied to these elements: <input>, <textarea>, <select> , and <button> .

Example:

```
<input type="text" name="fname" autofocus />
```

3.11 autocomplete

The autocomplete attribute allows the browsers to:

- predict the value when users start typing into an input field
- and display options to fill in the field based on earlier user input.

It works with <form> and these input types: text, password, email , and url .

Example:

```
<input type="text" name="fname" autocomplete />
```

Form Validation

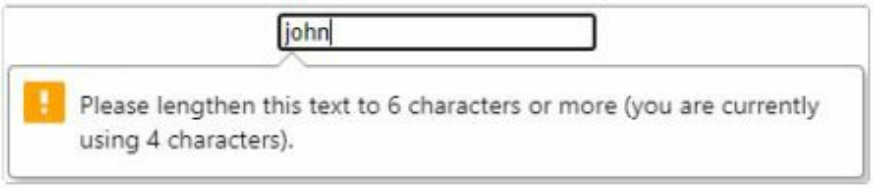
4.1 minlength

The minlength attribute specifies the minimum number of characters required in an input field. This attribute works with <input> (input types: text, password, email , and url) and <textarea> .

Example:

```
<input type="text" name="username" minlength="6" />
```

Result:



4.2 maxlength

The maxlength attribute specifies the maximum number of characters allowed in an input field. This attribute works with <input> (input types: text, password, email , and url) and <textarea> .

Example:

```
<input type="text" name="username" maxlength="6" />
```

In the above example, users cannot enter the 7^{th} character into the input.

4.3 min and max

The min attribute specifies the minimum value of <input> and <meter> elements. On the other hand, the max attribute specifies the maximum value of those elements.

The attributes work with these input types: number, range, date, time , and datetime-local .

Example:

```
<input type="number" name="quantity" min="1" max="8" />
```

4.4 required

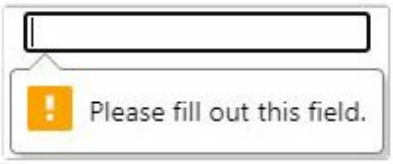
The required attribute specifies that an input field must be entered as a value before submitting the form. This attribute works with these elements: <input>, <textarea> , and <select> .

 The Safari browser does not support the required attribute.

Example:

```
<input type="text" name="fname" required />
```

Result:



4.5 pattern

The pattern attribute specifies a regular expression that the element's value will be checked against on form submission. This attribute works with these input types: text, password, email , and url .

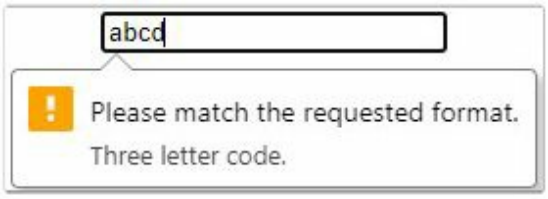
The “ title ” attribute of the input will be used as a hint when validation fails.

 The Safari browser does not support the pattern attribute.

Example:

```
<input type="text" name="code" pattern="[A-Za-z]{3}" title="Three letter code." />
```

Result:



4.6 Styles for The Invalid Inputs

Some CSS pseudo-classes help us highlight the invalid inputs before submitting the form.

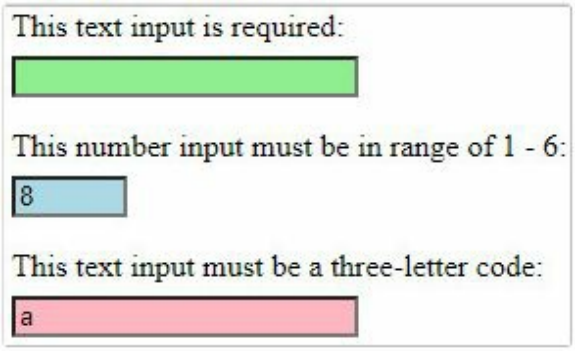
- " :invalid " is used to select form elements if their values are not legal according to the element's settings.
- " :required " is used to select required form elements.
- " :out-of-range " is used to select form elements with a value outside a specified range according to the min and max attributes.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <style>
   p {
    margin-bottom: 6px;
   }
   input:invalid {
    background-color: lightpink;
   }
   input:invalid:required {
    background-color: lightgreen;
   }
   input:out-of-range {
    background-color: lightblue;
   }
  </style>
 </head>
 <body>
  <form action="">
   <p>This text input is required:</p>
   <input type="text" name="fname" required />
```

```
    <p>This number input must be in the range of 1 - 6:</p>
    <input type="number" name="quantity" min="1" max="6" />

    <p>This text input must be a three-letter code:</p>
    <input type="text" name="code" pattern="[A-Za-z]{3}" title="Three letter code." />
   </form>
  </body>
</html>
```

Result:



CHAPTER 6
HTML Multimedia

Audio

We use the <audio> elements to embed audio files into our webpage. These are supported audio formats: MP3, WAV, and OGG but MP3 is recommended since almost browsers support it.

Attributes of the <audio> element:

- controls - shows the control panel (play/pause, timeline, and volume)
- autoplay - plays the audio right after the page loaded
- muted – no sound mode
- loop - auto play the audio again when playing done
- preload - the audio file should be loaded when the page loads. The preload attribute is ignored if autoplay is present
- src - specifies the location (URL) of the audio file

Instead of using the " src " attribute, <audio> could have some <source> elements as its children to specify the audio files. Then, browsers will choose the best audio source they can handle and play it.

Example 1:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <audio controls autoplay loop>
   <source src="my-music.ogg" type="audio/ogg" />
   <source src="my-music.mp3" type="audio/mpeg" />
   This browser does not support the audio element.
```

```
  </audio>
 </body>
</html>
```

Example 2:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <audio controls autoplay loop src="my-music.mp3"></audio>
 </body>
</html>
```

Result:

Video

We use the <video> elements to embedded video files into our webpage. These are supported video formats: MP4, WebM, and Ogg, but MP4 is recommended since almost browsers support it.

Attributes of the <video> element:

- controls - shows the control panel (play/pause, timeline, and volume)
- autoplay - plays the audio right after the page loaded
- muted – no sound mode
- loop - auto play the audio again when playing done
- preload - the audio file should be loaded when the page loads. The preload attribute is ignored if autoplay is present
- src - specifies the location (URL) of the audio file.
- width - width of the video player
- height - height of the video player

Instead of using the " src " attribute, <video> could have some <source> elements as its children to specify the video files. Then, browsers will choose the best video source they can handle and play it.

Example 1:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <video src="my-video.mp4" controls autoplay loop></video>
  </body>
</html>
```

Example 2:

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

```
  <title>My Website</title>
 </head>
 <body>
  <video width="320" height="240" autoplay>
   <source src="my-video.mp4" type="video/mp4" />
   <source src="my-video.ogg" type="video/ogg" />
   This browser does not support the video element.
  </video>
 </body>
</html>
```

CHAPTER 7
HTML Graphics

SVG Graphics

SVG stands for Scalable Vector Graphics and defines graphics for the web. SVG images are zoomed or resized without losing quality, often used as icons on web pages. Moreover, SVG files are pure XML, so they can be compressed before transferring via the internet.

We use <svg> element to define an SVG image. The “ width ” and “ height ” attributes determine the image’s width and height. This element often has several children elements to define/draw the picture, like the below example having a <line> element.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <svg width="100" height="100">
   <line x1="0" y1="0" x2="100" y2="50" stroke="red" stroke-width="2" />
  </svg>
 </body>
</html>
```

1.1 SVG Rectangle

We use the <rect> element to draw a rectangle in an SVG image or a <svg> tag. Its top-left corner will be at point (x, y), and its width and height are defined by the “ width ” and “ height ” attributes.

In the case of a shape like this, we use the “ fill ” attribute to specify its filled color. On the other hand, if we need only the outline, we can set “ none ” to the “ fill ” attribute.

Example:

```
<svg width="100" height="100">
  <rect x="10" y="10" width="80" height="60" stroke="red" stroke-width="2" fill="yellow" />
</svg>
```

Result:

We can draw a rounded rectangle using the “ rx ,” and “ ry ” attributes like this.

```
<svg width="100" height="100">
  <rect x="10" y="10" width="80" height="60" rx="8" ry="8" stroke="red" stroke-width="2"
fill="yellow" />
</svg>
```

Result:

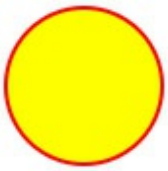
1.2 SVG Circle

We use the <circle> element to draw a circle in an SVG image or a <svg> tag. Its center will be at point (cx, cy), and the “ r ” attribute defines its radius.

Example:

```
<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="red" stroke-width="2" fill="yellow" />
</svg>
```

Result:



1.3 SVG Ellipse

We use the <ellipse> element to draw an ellipse in an SVG image or a <svg> tag. Its center will be at point (cx, cy), and the “ rx ” attribute defines its horizontal radius; the “ ry ” attribute defines its vertical radius.

Example:

```
<svg width="100" height="100">
  <ellipse cx="50" cy="50" rx="40" ry="20" fill="yellow" stroke="red" stroke-width="2" />
</svg>
```

Result:

1.4 SVG Line

We use the <line> element to draw a line in an SVG image or a <svg> tag. The line will start from a point (x1, y1) and be drawn to a point (x2, y2). Finally, we must specify the stroke color using the " stroke " attribute; nothing will be drawn without it!

Example:

```
<svg width="100" height="100">
  <line x1="0" y1="0" x2="100" y2="50" stroke="red" stroke-width="2" />
</svg>
```

Result:

1.5 SVG Polyline

We use the <polyline> element to draw a shape that consists of many connected straight lines in an SVG image or a <svg> tag. Its " points " attribute defines a list of points (x, y coordinates) required to draw the polyline.

Example:

```
<svg width="100" height="100">
  <polyline points="50,0 100,50 50,100 0,50" fill="none" stroke="red" stroke-width="2" />
</svg>
```

Result:

1.6 SVG Polygon

The <polygon> element works like an auto “closed” <polyline> . It means that a line will automatically connect the final point to the first point.

Example:

```
<svg width="100" height="100">
  <polygon points="50,0 100,50 50,100 0,50" fill="white" stroke="red" stroke-width="2" />
</svg>
```

Result:

1.7 SVG Path

The <path> element defines a path of lines and curves. Path data, the “ d ” attribute can use these commands to describe a path:

- M = move the pen to
- L = draw a line to
- H = draw a horizontal line to
- V = draw a vertical line to
- C = draw a curve to
- S = draw a smooth curve to
- Q = draw a quadratic Bézier curve
- T = draw a smooth quadratic Bézier curve to
- A = draw an elliptical Arc
- Z = draw a line to close the path

The capital letter means positioned to the point (0,0), and lower-case letter means relatively positioned to the previous point in the path.

Example:

```
<svg height="100" width="100">
```

```
  <path d="M50 0 L25 80 l50 0 Z" fill="none" stroke="red" stroke-width="2" />
</svg>
```

Result:

Explanation:

- M50 0 - move the pen to point P1(x1, y1) = (50, 0)
- L25 80 - draw a line from the previous point (P1) to the point P2(x2, y2) = (25, 80)
- l50 0 - draw a line from the previous point (P2) to the point P3(x2 + 50, y2 + 0) = (25 + 50, 80 + 0) = (75, 80)
- Z – draw a line from the previous point (P3) to the first point (P1)

1.8 SVG Text

The <text> element defines a text.

Example:

```
<svg height="100" width="200">
  <text x="0" y="20" fill="red">This is a line of text.</text>
</svg>
```

The <text> element can handle a group of <tspan> elements. Each <tspan> element works similarly to <text> .

Example:

```
<svg height="100" width="200">
  <text x="0" y="20" fill="red">
    This is a line of text.
    <tspan x="10" y="50" fill="blue">This is the second line.</tspan>
    <tspan x="20" y="80">This is the third line.</tspan>
  </text>
</svg>
```

Moreover, we can transform the text using the “ transform ” attribute like this.

```
<svg height="100" width="200">
  <text x="20" y="20" fill="red" transform="rotate(30)">This is a line of text.</text>
</svg>
```

1.9 SVG Link

We use an <a> element to “draw” a hyperlink in an SVG image. The link text is defined by using a <text> element.

Example:

```
<svg height="100" width="200" xmlns:xlink="http://www.w3.org/1999/xlink">
  <a xlink:href="https://www.amazon.com/dp/B09VFLS7TF" target="_blank">
    <text x="0" y="20" fill="blue">This is a link</text>
  </a>
</svg>
```

1.10 SVG Stroke

The below stroke properties can be applied to text, lines, and outlines of shapes (rectangles, circles, etc.)

1.10.1 Stroke Color

We use the “ stroke ” attribute to define the color of a stroke.

Example:

```
<svg width="100" height="100">
  <line x1="0" y1="0" x2="100" y2="50" stroke="red" stroke-width="2" />
</svg>
```

Result:

1.10.2 Stroke Width

We use the “ stroke-width ” attribute to define the thickness of a stroke like the above example.

1.10.3 Stroke Ending

We use the “ stroke-linecap ” attribute to define the type of endings of an open path. It can have one of these values:

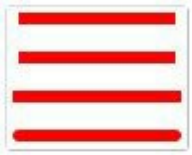
- butt - no endings (default value)
- square - defines square endings
- round - defines round endings

Example:

```
<svg width="100" height="100">
  <line x1="10" y1="10" x2="90" y2="10" stroke="red" stroke-width="6" />
  <line x1="10" y1="30" x2="90" y2="30" stroke="red" stroke-width="6" stroke-linecap="butt" />
  <line x1="10" y1="50" x2="90" y2="50" stroke="red" stroke-width="6" stroke-linecap="square"
/>
  <line x1="10" y1="70" x2="90" y2="70" stroke="red" stroke-width="6" stroke-linecap="round"
```

```
/>
</svg>
```

Result:



In the above example, square and round endings make the line longer than its actual length, more than 80px.

1.10.4 Dash Stroke

We use the " stroke-dasharray " attribute to create dashed lines. The odd items in the dash array will be used to draw the line segments, and the even items specify the spaces between them.

Example:

```
<svg width="300" height="100">
  <line x1="0" y1="10" x2="300" y2="10" stroke="red" stroke-width="2" stroke-
dasharray="10,5,20" />
</svg>
```

Result:

Explanation:

- 10 was used to draw the 1st line segment
- 5 was used to make the 1st space
- 20 was used to draw the 2nd line segment

since the line is 300px in length, it continues to use the dash array to draw the rest

- 10 was used to make the 2nd space
- 5 was used to draw the 3rd line segment
- 20 was used to make the 3rd space

and

- 10 was used to draw the 4th line segment
- 5 was used to make the 4th space
- 20 was used to draw the 5th line segment

and so on.

1.11 SVG Gradients

1.11.1 Linear Gradient

The <linearGradient> element defines a linear gradient for filling a shape in an SVG image. It is placed within a <defs> tag (short for "definitions" since the tag can contain many definitions). The gradient is composed of two or more colors. Each color is defined with a <stop> nested within the <linearGradient> tag.

Finally, the (x1,y1) and (x2,y2) attributes of the <linearGradient> element define the start and end position of the gradient. Thus,

- if x1 and x2 are different and y1 and y2 are equal, the gradient is horizontal
- if x1 and x2 are equal and y1 and y2 are different, the gradient is vertical
- if x1 and x2 are different and y1 and y2 are different, the gradient is vertical

Example 1:

```
<svg height="100" width="200">
  <defs>
    <linearGradient id="grad" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" stop-color="yellow" />
      <stop offset="50%" stop-color="green" />
      <stop offset="100%" stop-color="red" />
    </linearGradient>
  </defs>
  <rect x="0" y="0" width="200" height="100" fill="url(#grad)" />
</svg>
```

Result:

Example 2:

```
<svg height="100" width="200">
  <defs>
    <linearGradient id="grad" x1="0%" y1="0%" x2="100%" y2="100%">
      <stop offset="0%" stop-color="yellow" />
      <stop offset="50%" stop-color="green" />
      <stop offset="100%" stop-color="red" />
    </linearGradient>
  </defs>
  <rect x="0" y="0" width="200" height="100" fill="url(#grad)" />
</svg>
```

Result:

1.11.2 Radial Gradient

The <radialGradient> element defines a radial gradient for filling a shape in an SVG image. It is placed within a <defs> tag (short for “definitions” since the tag can contain many definitions). The gradient is composed of two or more colors. Each color is defined with a <stop> nested within the <linearGradient> tag.

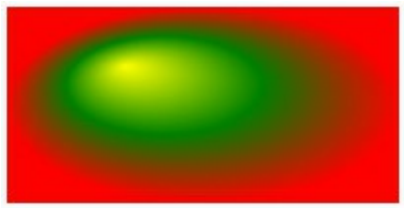
Finally, the (cx,cy) is the center, and r is the radius of the outermost circle; the (fx,fy) is the center of the innermost circle.

Example:

```
<svg height="100" width="200">
  <defs>
```

```
    <radialGradient id="grad" cx="50%" cy="50%" r="50%" fx="30%" fy="30%">
      <stop offset="0%" stop-color="yellow" />
      <stop offset="50%" stop-color="green" />
      <stop offset="100%" stop-color="red" />
    </radialGradient>
  </defs>
  <rect x="0" y="0" width="200" height="100" fill="url(#grad)" />
</svg>
```

Result:



1.12 SVG Filters

The <filter> element defines an SVG filter. There are many filters available in SVG, but we will only examine the Gaussian Blur, the <feGaussianBlur> element, in the example below.

The “ stdDeviation ” attribute of <feGaussianBlur> defines the amount of the blur, and the in="SourceGraphic" specifies that the effect is for the entire element.

Example:

```
<svg height="100" width="200">
  <defs>
    <filter id="my-filter">
      <feGaussianBlur in="SourceGraphic" stdDeviation="30" />
    </filter>
  </defs>
  <rect x="0" y="0" width="200" height="100" fill="green" filter="url(#my-filter)" />
</svg>
```

Result:

2 Canvas

The <canvas> element is used to draw graphics on a webpage via JavaScript. Therefore, it always has an “ id ” attribute to be later referred to by JavaScript code. Besides, the “ width ” and “ height ” attributes define the canvas size.

By default, the canvas has no border and a transparent background so that we can place it in front of another element, such as a video, and draw rectangles to mark things like cars on the road, for example.

Example:

```
<canvas id="my-canvas" width="200" height="100" style="border: 1px solid #ccc"> This browser does not support the canvas tag. </canvas>
```

Result:

2.1 Draw Text

To draw something, first, we must get the canvas element from our HTML document like this.

```
var c = document.getElementById("my-canvas");
```

Then, we need to get a drawing object from the canvas.

```
var ctx = c.getContext("2d");
```

Finally, we use the drawing object to draw text on our canvas using the fillText() method. We can also set a font for the text using the “ font ” property.

Example:

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <canvas id="my-canvas" width="200" height="100" style="border: 1px solid #ccc"> This
browser does not support the canvas tag. </canvas>

    <script>
      var c = document.getElementById("my-canvas");
      var ctx = c.getContext("2d");
      ctx.font = "20px Arial";
      ctx.fillText("Neo D. Truman", 10, 50);
    </script>
  </body>
</html>
```

Result:

2.2 Draw Lines

We also get the drawing object from our canvas to draw a line. Then use these three methods to draw it:

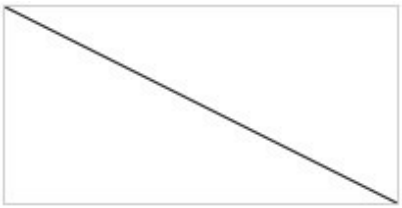
- moveTo(x, y) - move pen to the point (x,y)
- lineTo(x, y) - draw a line from current position to the point (x,y)
- stroke() - invoke the drawing

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <canvas id="my-canvas" width="200" height="100" style="border: 1px solid #ccc"> This
browser does not support the canvas tag. </canvas>
```

```
    <script>
      var c = document.getElementById("my-canvas");
      var ctx = c.getContext("2d");
      ctx.moveTo(0, 0);
      ctx.lineTo(200, 100);
      ctx.stroke();
    </script>
  </body>
</html>
```

Result:



2.3 Draw Circles

We also get the drawing object from our canvas to draw a circle. Then use these two methods to draw it:

- arc(x,y,r,startangle,endangle) - draw an arc with the center point at (x,y) , and the radius is r , from start angle to end angle
- stroke() - invoke the drawing

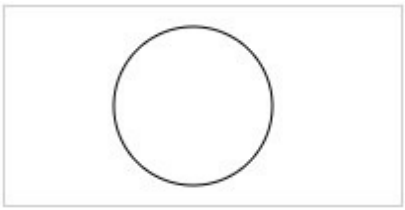
Since we want to draw a circle, the start angle is set to 0 and the end angle to 2*PI.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <canvas id="my-canvas" width="200" height="100" style="border: 1px solid #ccc"> This
browser does not support the canvas tag. </canvas>

    <script>
      var c = document.getElementById("my-canvas");
```

```
    var ctx = c.getContext("2d");
    ctx.arc(95, 50, 40, 0, 2 * Math.PI);
    ctx.stroke();
  </script>
 </body>
</html>
```

Result:



2.4 Draw Rectangles

We also get the drawing object from our canvas to draw a rectangle. Then use these two methods to draw it:

- rect(x,y,width,height) - draw a rectangle with the top-left corner at point (x,y) and width and height as specified
- stroke() - invoke the drawing

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <canvas id="my-canvas" width="200" height="100" style="border: 1px solid #ccc"> This browser does not support the canvas tag. </canvas>

    <script>
      var c = document.getElementById("my-canvas");
      var ctx = c.getContext("2d");
      ctx.rect(20, 20, 160, 60);
      ctx.stroke();
    </script>
  </body>
</html>
```

Result:

2.5 Draw Gradients

2.5.1 Linear Gradient

We use the createLinearGradient(x1,y1,x2,y2) method to create a linear

gradient where (x1,y1) and (x2,y2) define the start and end position of the gradient. Besides, the addColorStop() method specifies a color stop along the gradient; its value can be anywhere between 0 to 1.

Finally, we use the " fillStyle " property to specify the gradient to be filled in the shape.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <canvas id="my-canvas" width="200" height="100" style="border: 1px solid #ccc"> This browser does not support the canvas tag. </canvas>

    <script>
      var c = document.getElementById("my-canvas");
      var ctx = c.getContext("2d");

      // Create gradient
      var grd = ctx.createLinearGradient(0, 0, 200, 0);
      grd.addColorStop(0, "red");
      grd.addColorStop(1, "yellow");
      // Fill with gradient
      ctx.fillStyle = grd;

      // Draw a filled rect
      ctx.fillRect(20, 20, 160, 60);
    </script>
  </body>
</html>
```

Result:

2.5.2 Radial Gradient

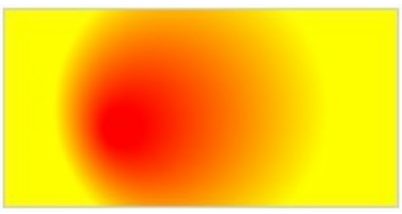
We use the createRadialGradient(x1,y1,r1,x2,y2,r2) method to create a radial gradient where (x1,y1) is the center and r1 is the radius of the innermost circle; the (x2,y2) is the center, and r2 is the radius of the outermost circle. Besides, the addColorStop() method specifies a color stop along the gradient; its value can be anywhere between 0 to 1.

Finally, we use the " fillStyle " property to specify the gradient to be filled in the shape.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <canvas id="my-canvas" width="200" height="100" style="border: 1px solid #ccc"> This
browser does not support the canvas tag. </canvas>

    <script>
      var c = document.getElementById("my-canvas");
      var ctx = c.getContext("2d");

      // Create gradient
      var grd = ctx.createRadialGradient(60, 60, 10, 95, 50, 70);
      grd.addColorStop(0, "red");
      grd.addColorStop(1, "yellow");
      // Fill with gradient
      ctx.fillStyle = grd;

      // Draw a filled rect
      ctx.fillRect(0, 0, 200, 100);
    </script>
  </body>
</html>
```

Result:



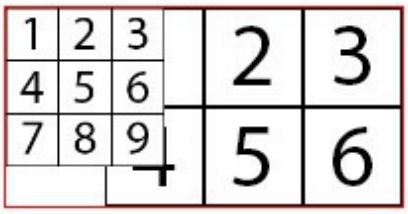
2.6 Draw Images

We use the drawImage(image,x,y,width,height) method to draw an image on canvas at the point (x,y) . If width and height parameters are not specified, the intrinsic size of the image will be used.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <canvas id="my-canvas" width="200" height="100" style="border: 1px solid red"> This browser does not support the canvas tag. </canvas>

    <script>
      var c = document.getElementById("my-canvas");
      var ctx = c.getContext("2d");

      const image = new Image();
      // Set a callback function when the image has been loaded
      image.onload = drawMyImage;

      // Load an image of intrinsic size 150x150 in pixels
      image.src = "my-img.jpg";

      function drawMyImage() {
        // At point (50,0), draw the image at its intrinsic size 150x150
        ctx.drawImage(this, 50, 0);

        // At point (0,0), draw the image at the size 80x80
        ctx.drawImage(this, 0, 0, 80, 80);
      }
    </script>
  </body>
</html>
```

Result:


1 2 3
4 5 6
7 8 9
2 3
5 6

CHAPTER 8
HTML Advanced

Data URIs

Data URI (**U**niform **R**esource **I**dentifier) supports including inline data in web pages as if they were external resources. Below is the syntax of a data URI:

```
data:[<media type>][;base64],<data>
```

Where:

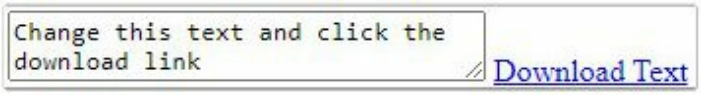
- <media type> - (optional) is a MIME type string having this format: type/subtype;parameter=value . For example, if it is omitted, the default value is " text/plain;charset=US-ASCII "
- ;base64 - (optional) specifies that the data is base64-encoded binary data, so it is used with media types like images.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <textarea name="input" cols="30">Change this text and click the download link</textarea>

  <a href="javascript:void(0)" download="data.txt"> Download Text </a>

  <script type="text/javascript">
   var input = document.querySelector("textarea[ name = 'input' ]");
   var download = document.querySelector("a[ download ]");

   // Listen for input changes so that we can update the HREF
   // attribute of the download link to contain the proper Data URI
```

```
    input.addEventListener("input", updateDownloadHref, false);

    // Initialize the download link
    updateDownloadHref();

    function updateDownloadHref() {
      var text = input.value;
      // Build the Data URI
      download.setAttribute("href", "data:text/plain;charset=utf-8," + encodeURIComponent(text));
    }
   </script>
  </body>
</html>
```

Result:



2 Block Elements

Block elements occupy 100% of the parent element's width, regardless of content. Therefore, they are stacked vertically by default.

These elements could have more space than their contents by using these CSS properties: width, height, padding, and margin .

These are HTML block elements: body, header, nav, main, section, footer, h1-h6, p, div, ul, ol, li, pre, blockquote, address, details, figure, fieldset , etc.

We can change other types of elements to block elements using this CSS declaration:

```
display: block
```

3 Inline Elements

Inline elements only occupy the space necessary for their contents so that they will not cause any line breaks. Therefore, they could not use these CSS properties: width, height, padding-top, padding-bottom, margin-top, margin-bottom .

These are HTML inline elements: formatting elements (b, strong, i, em, mark, etc.), label, a, span, code, img, button , etc.

We can change other types of elements to inline elements using this CSS declaration:

```
display: inline
```

Inline-block Elements

The inline-block elements have both characteristics of inline and block elements. They only occupy the space necessary for their contents so that they will not cause any line breaks. However, they could have more space than their contents by using these CSS properties: width, height, padding, and margin .

These are HTML inline-block elements: button, input, textarea, select, progress, meter, iframe, img, svg, canvas, audio, video , etc.

We can change other types of elements to inline-block elements using this CSS declaration:

```
display: inline-block
```

CHAPTER 9
HTML Events

HTML can let events trigger actions in a browser, like invoking a JavaScript function when users click on an element.

. Window Events

1.1 onload

The “ onload ” attribute fired when the browser finished loading a webpage. We usually handle the “ onload ” event on these elements: <body>, <iframe>, <img>, <link> , and <script> .

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <script>
      function eventHandler() {
        alert("Page is loaded");
      }
    </script>
  </head>
  <body onload="eventHandler()">
    <h1>This is a heading</h1>
  </body>
</html>
```

1.2 onresize

The “ onresize ” attribute fires when the browser window is resized.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <script>
      function eventHandler() {
        alert("You have changed the size of your browser window!");
      }
    </script>
  </head>
  <body onresize="eventHandler()">
    <h1>This is a heading</h1>
  </body>
</html>
```

2 Form Events

2.1 onfocus

The “ onfocus ” attribute fires when the element gets focused.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  First name: <input type="text" onfocus="eventHandler(this)" />

  <script>
   function eventHandler(input) {
    input.style.background = "yellow";
   }
  </script>
 </body>
</html>
```

2.2 onblur

The “ onblur ” attribute fires when the element loses focus.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  Enter your name: <input type="text" id="fname" onblur="eventHandler()" />

  <script>
   function eventHandler() {
    var x = document.getElementById("fname");
    alert("Hi " + x.value);
   }
```

```
  </script>
 </body>
</html>
```

2.3 onchange

The “ onchange ” attribute fires when the element’s value has been changed. Supported HTML tags: <input type="checkbox">, <input type="file">, <input type="password">, <input type="radio">, <input type="range">, <input type="text">, <select> and <textarea> .

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <select id="my-select" onchange="eventHandler()">
   <option value="Audi">Audi</option>
   <option value="BMW">BMW</option>
   <option value="Mercedes">Mercedes</option>
   <option value="Volvo">Volvo</option>
  </select>

  <script>
   function eventHandler() {
    var x = document.getElementById("my-select").value;
    alert("You selected: " + x);
   }
  </script>
 </body>
</html>
```

2.4 oninput

The “ oninput ” attribute fires when an element gets user input.

Example:

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

```
    <title>My Website</title>
  </head>
  <body>
    <p>Write some text into the box!</p>

    <input type="text" id="my-input" oninput="eventHandler()" />

    <p id="output"></p>

    <script>
      function eventHandler() {
        var x = document.getElementById("my-input").value;
        document.getElementById("output").innerHTML = "You wrote: " + x;
      }
    </script>
  </body>
</html>
```

3 Keyboard Events

3.1 onkeydown

The “ onkeydown ” attribute fires when users press a key on the keyboard.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>A function is triggered when the user presses a key in the input field.</p>

    <input type="text" onkeydown="eventHandler()" />

    <script>
      function eventHandler() {
        alert("You pressed a key inside the input field");
      }
    </script>
  </body>
</html>
```

3.2 onkeyup

The “ onkeyup ” attribute fires when the users release a key on the keyboard.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>A function is triggered when the user releases a key in the input field. This function
transforms the character to upper case.</p>
```

```
  Enter your name: <input type="text" id="fname" onkeyup="eventHandler()" />

  <script>
    function eventHandler() {
      var x = document.getElementById("fname");
      x.value = x.value.toUpperCase();
    }
  </script>
 </body>
</html>
```

3.3 onkeypress

The “ onkeypress ” attribute fires when users press a key on the keyboard.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>A function is triggered when the user presses a key in the input field.</p>

  <input type="text" onkeypress="eventHandler()" />

  <script>
    function eventHandler() {
      alert("You pressed a key inside the input field");
    }
  </script>
 </body>
</html>
```

Mouse Events

4.1 onclick

The “ onclick ” attribute fires on a mouse click on the element.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <button onclick="eventHandler()">Click me</button>

    <script>
      function eventHandler() {
        alert("You clicked the button");
      }
    </script>
  </body>
</html>
```

4.2 ondblclick

The “ ondblclick ” attribute fires on a mouse double-click on the element.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <button ondblclick="eventHandler()">Click me</button>

    <script>
      function eventHandler() {
        alert("You double-clicked the button");
      }
    </script>
  </body>
</html>
```

4.3 onmousemove

The “ onmousemove ” attribute fires when the pointer is moving while it is over an element.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p onmousemove="enlarge(this)">The function enlarge() is triggered when the user mouse pointer is moved over the paragraph. This function will enlarge the font size.</p>

  <script>
   function enlarge(x) {
    var oldFontSize = x.style["font-size"];
    if (!oldFontSize) {
     oldFontSize = 16;
    } else {
     oldFontSize = parseFloat(oldFontSize.replace("px", ""));
    }

    x.style["font-size"] = oldFontSize + 0.1 + "px";
   }
  </script>
 </body>
</html>
```

4.4 onmousedown and onmouseup

The “ onmousedown ” attribute fires when a mouse button is pressed on the element. On the other hand, the “ onmouseup ” attribute fires when a mouse button is released over the element.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p id="p1" onmousedown="mouseDown()" onmouseup="mouseUp()">
   Click this text!<br />
   The mouseDown() function is triggered when the mouse button is pressed over this paragraph. The function sets the color of the text to red. <br />
   The mouseUp() function is triggered when the mouse button is released while over the paragraph. The function sets the color of the text to blue.
  </p>

  <script>
   function mouseDown() {
    document.getElementById("p1").style.color = "red";
   }

   function mouseUp() {
    document.getElementById("p1").style.color = "blue";
   }
  </script>
 </body>
</html>
```

4.5 onmouseover and onmouseout

The “ onmouseover ” attribute fires when the mouse pointer moves over an element. On the other hand, the “ onmouseout ” attribute fires when the mouse pointer moves out of an element.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p onmouseover="increaseFontSize(this)" onmouseout="decreaseFontSize(this)">
   When the user mouse over the paragraph, its font size will be 32px. <br />
   When the mouse pointer is moved out of it, the font size will be 20px.
  </p>

  <script>
   function increaseFontSize(x) {
    x.style["font-size"] = "32px";
   }

   function decreaseFontSize(x) {
    x.style["font-size"] = "20px";
   }
  </script>
 </body>
</html>
```

5 Clipboard Events

5.1 oncopy

The “ oncopy ” attribute fires when the user copies the content of an element. There are two ways to copy the content:

- Press CTRL + C
- Right-click to display the context menu and select the "Copy" command

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <input type="text" oncopy="eventHandler()" value="Try to copy this text" />

    <script>
      function eventHandler() {
        alert("You copied the text.");
      }
    </script>
  </body>
</html>
```

5.2 oncut

The “ oncut ” attribute fires when the user cuts the content of an element. There are two ways to cut the content:

- Press CTRL + X
- Right-click to display the context menu and select the "Cut" command

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <input type="text" oncut="eventHandler()" value="Try to cut this text" />

    <script>
      function eventHandler() {
        alert("You cut the text.");
      }
    </script>
  </body>
</html>
```

5.3 onpaste

The “ onpaste ” attribute fires when the user pastes some content in an element. There are two ways to paste some content into an element:

- Press CTRL + V
- Right-click to display the context menu and select the "Paste" command

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <input type="text" onpaste="eventHandler()" value="Try to paste text in here" size="40" />

    <script>
      function eventHandler() {
        alert("You pasted the text.");
      }
    </script>
  </body>
</html>
```

CHAPTER 10
HTML APIs

.1 Drag and Drop

Using JavaScript code, we can drag and drop an HTML element into another element.

First of all, to make an element draggable, set its “ draggable ” attribute to true:

```
<div id="div1" draggable="true">This div is draggable</div>
```

Then, specify what happens when the element is dragged using the “ ondragstart ” attribute. Finally, we will define a callback function for it to invoke when users start dragging. In this function, we will call the dataTransfer.setData() method to set the dragged data.

```
<div id="div1" draggable="true" ondragstart="dragStartHandler(event)">This div is
draggable</div>

<script>
  function dragStartHandler(evt) {
    evt.dataTransfer.setData("sourceID", evt.target.id);
  }
</script>
```

By default, HTML elements do not allow dropping other elements onto them. So, to enable it, we have to prevent the default handling of those HTML elements by using the “ ondragover ” attribute like the below code.

```
<div id="div2" ondragover="allowDropping(event)"></div>

<script>
  function allowDropping(evt) {
    evt.preventDefault();
  }
```

```
</script>
```

When the dragged element is dropped, a drop event occurs, so the callback function setting at the “ ondrop ” attribute will be called. First, the callback function will get the dragged data (the dragged element’s ID) using the dataTransfer.getData() method. Then, it gets the element by the ID and appends it to the destination element.

```
<div id="div2" ondrop="dropHandler(event)" ondragover="allowDropping(event)"></div>

<script>
  function dropHandler(ev) {
    var data = ev.dataTransfer.getData("sourceID");
    ev.target.appendChild(document.getElementById(data));
  }
</script>
```

This is the final code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <style>
      #div2 {
        border: 1px solid black;
        width: 200px;
        height: 60px;
      }
    </style>
  </head>
  <body>
    <div id="div1" draggable="true" ondragstart="dragStartHandler(event)">This div is draggable.
You can drag and drop it into the rectangle.</div>

    <div id="div2" ondrop="dropHandler(event)" ondragover="allowDropping(event)"></div>

    <script>
      function allowDropping(ev) {
        ev.preventDefault();
      }

      function dropHandler(ev) {
        var data = ev.dataTransfer.getData("sourceID");
```

```
      ev.target.appendChild(document.getElementById(data));
    }

    function dragStartHandler(ev) {
      console.log(ev);
      ev.dataTransfer.setData("sourceID", ev.target.id);
    }
  </script>
 </body>
</html>
```

.2 Geolocation

The HTML Geolocation API in JavaScript helps us get the user's geographical position (latitude and longitude). By default, the position is unavailable unless the user approves it because this can compromise privacy.

We use geolocation.getCurrentPosition() method of the navigator object to get the user's current position. The function receives a callback as its parameter to return a coordinate object to the callback.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>Click the button to get your coordinates.</p>

  <button onclick="getLocation()">Get location</button>

  <p id="out"></p>

  <script>
   var x = document.getElementById("out");

   function getLocation() {
    if (navigator.geolocation) {
     navigator.geolocation.getCurrentPosition(showPosition);
    } else {
     x.innerHTML = "Geolocation is not supported by this browser.";
    }
   }

   function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude + "<br>Longitude: " +
position.coords.longitude;
   }
  </script>
 </body>
</html>
```

.3 Web Storage

Before HTML5, application data had to be stored in cookies which would be included in every server request. Consequently, using cookies affects our website performance.

Our web applications can store data locally within the browsers with web storage. Web storage is more secure, large amounts of data (at least 5MB) can be stored locally, and it is never transferred to the server.

Before using web storage, we should check browser support for it:

```
if (typeof Storage !== "undefined") {
  // Code for Web Storage
} else {
  // Not support Web Storage
}
```

HTML web storage provides two objects for storing data on the client:

- localStorage - stores data with no expiration date
- sessionStorage - stores data for one session (data is lost when the browser tab is closed)

.3.1 The localStorage Object

The localStorage object stores the data with no expiration date and will not be deleted when the browser is closed.

We can store and retrieve the data using setItem() and getItem() methods like this example.

```
// Store
localStorage.setItem("firstname", "John");

// Retrieve
var firstname = localStorage.getItem("firstname");
```

The example above could also be written like this:

```
// Store
localStorage.firstname = "John";
```

```
// Retrieve
var firstname = localStorage.firstname;
```

The syntax for removing the “firstname” item in the local storage is as follows:

```
localStorage.removeItem("firstname");
```

Web storage data is always strings. Therefore, we must convert them to another format when needed.

Let’s examine the below example to see how Local Storage works in reality.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>Step 1. Enter your name:</p>
    <input id="name" type="text" />

    <p>Step 2. Click this button to set it to the Local Storage:</p>
    <input type="button" value="Set data" onclick="setItem()" />

    <p>Step 3. Open this webpage in another tab.</p>

    <p>Step 4. Click this button to get data from the Local Storage:</p>
    <input type="button" value="Get data" onclick="getItem()" />

    <p>Step 5. Close all tabs and open this page again. Then, click the "Get data" button to see if we can still get the data.</p>

    <script>
      function setItem() {
        if (typeof Storage !== "undefined") {
          var name = document.getElementById("name").value;
          localStorage.myName = name;
        }
```

```
    }

    function getItem() {
      if (typeof Storage !== "undefined") {
        alert(localStorage.myName);
      }
    }
  </script>
 </body>
</html>
```

.3.2 The sessionStorage Object

The sessionStorage object is similar to the localStorage object, except that it stores the data for only one session. It means that the data will be deleted when users close the tab on the browser.

Example:

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
 </head>
 <body>
  <p>Step 1. Enter your name:</p>
  <input id="name" type="text" />

  <p>Step 2. Click this button to set it to the Session Storage:</p>
  <input type="button" value="Set data" onclick="setItem()" />

  <p>Step 3. Click this button to get data from the Session Storage:</p>
  <input type="button" value="Get data" onclick="getItem()" />

  <p>Step 4. Open this webpage in another tab.</p>

  <p>Step 5. Then, click the "Get data" button to see that we cannot get the data since it's only
available in the first tab.</p>

  <script>
    function setItem() {
      if (typeof Storage !== "undefined") {
        var name = document.getElementById("name").value;
        sessionStorage.myName = name;
```

```
    }
  }

  function getItem() {
    if (typeof Storage !== "undefined") {
      alert(sessionStorage.myName);
    }
  }
 </script>
</body>
</html>
```

.4 Web Workers

The page becomes unresponsive when executing JavaScript code on a webpage until the execution is finished. Therefore, when we have JavaScript code that could run for a long time, we put it in a Web Worker.

A Web Worker is a JavaScript code running in the background without blocking users from doing whatever they want on the webpage, such as clicking, selecting things, etc.

.4.1 Web Worker File

Let's create our web worker in a separate JavaScript file. The script will increase a counter every second.

my-web-worker.js

```
var i = 0;

function startCounter() {
  i = i + 1;
  postMessage(i);
  setTimeout("startCounter()", 1000);
}

startCounter();
```

The most critical line of the above code is the postMessage() method which is used to post a message back to our HTML page.

.4.2 Web Worker Object

Now, from our HTML page, let’s create a web worker object and use it to receive messages from the worker. However, before creating it, we should check whether the web browser supports it:

```
if (typeof Worker !== "undefined") {
  // Yes, Web Worker is supported.
  // Your code here...
} else {
  // No, Web Worker is not supported!
```

```
}
```

Then, we create a Web Worker object

```
w = new Worker("my-web-worker.js");
```

and receive messages from the web worker by adding an “ onmessage ” event listener to the web worker object.

```
w.onmessage = function (event) {
  document.getElementById("result").innerHTML = event.data;
};
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data .

Finally, to terminate a web worker and free the resources of the browser, we use the terminate() method and set the variable to undefined :

```
w.terminate();
w = undefined;
```

We should stop/terminate the worker when finished. Otherwise, it will continue to listen for messages (even after the external worker script is done) until it is terminated.

Below is the code for the HTML page:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>Count numbers: <output id="result"></output></p>
    <button onclick="startWorker()">Start Worker</button>
    <button onclick="stopWorker()">Stop Worker</button>

    <script>
      var w;

      function startWorker() {
        if (typeof Worker !== "undefined") {
```

```
        if (typeof w == "undefined") {
          w = new Worker("my-web-worker.js");
        }
        w.onmessage = function (event) {
          document.getElementById("result").innerHTML = event.data;
        };
      } else {
        document.getElementById("result").innerHTML = "This browser does not support Web
Workers!";
      }
    }

    function stopWorker() {
      w.terminate();
      w = undefined;
    }
  </script>
 </body>
</html>
```

9.4.3 Web Workers and the DOM

Since web workers are placed in external JavaScript files and run in other threads, they do not have access to the following JavaScript objects of the DOM:

- The window object
- The document object

It means web workers cannot manipulate or interact with the HTML elements on our web pages.

Part II: CSS3

While talking about CSS3, we must mention its predecessor, CSS. CSS is an acronym for **C**ascading **S**tyle **S**heets, the language used to style web pages.

Why should we learn CSS3?

CSS3 is the 3rd and the latest version of CSS. CSS3 has added many more convenient features for users than CSS. Inheriting everything from the previous version and adding new features, CSS3 is now very popular in website design.

CSS is a tool that helps us add changes in appearance, such as changing layout, colors, fonts, etc. If a web page does not have CSS, it will simply be a page containing text with two primary colors, black and white.

CSS works by partitioning the selection based on the HTML tag name, ID, or class. From there, apply the properties to be changed to the selected HTML elements.

What will you learn?

In this part of the book, you will learn the following:

- Basic CSS: CSS syntax, comments, and units (absolute and relative lengths).
- Three types of CSS: Inline CSS, Internal CSS, and External CSS.
- CSS selectors: Universal selector, Type selector, ID selector, Class selector, Multiple-class selector, Pseudo-class selector, Pseudo-element, Combined selector, Attribute selector, and selector grouping.
- CSS cascading rules and inheritance in CSS.
- CSS box model: box-sizing types, margins, outline, borders, padding, width, and height.
- CSS colors: color names, HEX colors, RGB colors, and grey colors.
- CSS text: text rendering, text spacing, text position, CSS fonts.
- CSS backgrounds: background properties, gradient backgrounds, and CSS sprites.
- CSS display: display vs. visibility, object-fit, cursor, overflow, opacity, shadows, multiple columns, table styling, list styling, and CSS transforms.
- CSS layout: position, float, z-index, flexbox, grid, and CSS alignment.
- CSS transitions and animations.
- Advanced CSS: CSS variables, calculation, and CSS counters.
- Responsive CSS: responsive web design (RWD), RWD viewport, CSS media, RWD grid view, and Mobile-first vs. Desktop-first.

CHAPTER 11
Getting to Know CSS

CSS stands for **C**ascading **S**tyle **S**heets. It describes the visual style and the presentation of the HTML content in a web browser. It can also change a site's display based on different screen sizes and devices. In addition, CSS includes rules that format the content, such as font, text, layout, etc.

In this book, we will learn CSS3 since it is the latest version of CSS. CSS3 has added many more convenient features for users than CSS. Inheriting everything from the previous version and adding new features, CSS3 is now very popular in website design. However, I will write "CSS" as a shorthand for "CSS3" from now on.

.1 CSS Syntax

Let’s examine this example:

```
p {
  color: red;
  font-size: 28px;
}
```

The above CSS rule starts with a selector (“ p ”) followed by a declaration block (inside the curly brackets {}).

The declaration block includes all styles for elements matching the selector. A style declaration (e.g., “ color: red; ”) is a pair of a property (“ color ” in the example) and a value (“ red ” in the example) that describes the style. The semicolon (“ ; ”) separates many declarations.

.2 Comments in CSS

Any code should have comments to maintain the code later easily. The comments start with “ /* ” and end with “ */ .” Web browsers will ignore all comments.

Example:

```
/* Single Line */

/*
Multiple
Lines
*/
```

.3 CSS Units

CSS code has several different units for expressing a length. In addition, some CSS properties take length values, such as width, margin, padding, font-size, border-width , etc. A length is a number followed by a length unit, for example, 10px or 2em .

A whitespace cannot appear between the number and the unit. However, if the value is 0 , the unit can be omitted. Moreover, some CSS properties can have negative lengths.

..3.1 Absolute Lengths

These are the most popular absolute lengths:

- cm : centimeters
- mm : millimeters
- in : inches (1in = 96px = 2.54cm)
- px : pixels (1px = 1/96th of 1in)
- pt : points (1pt = 1/72 of 1in)

..3.2 Relative Lengths

Relative length units specify a length relative to another length property. They scale better between different screen sizes.

- rem : Relative to the font-size of the root element (2rem means two times the size of the root element's font)
- em : Relative to the font-size of the element (2em means two times the size of the current element's font)
- vw : Relative to 1% of the width of the viewport
- vh : Relative to 1% of the height of the viewport
- % : Relative to 1% of the containing element.

1.3.2.1 em

Example 1:

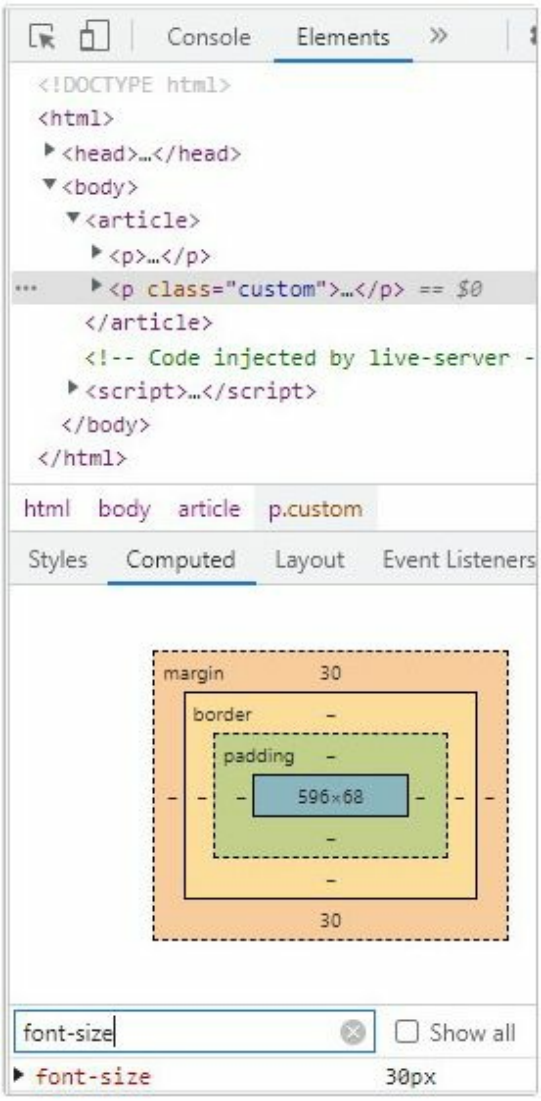
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <article>
      <p>
        This element does not have its style, so that the font-size will be 20px, inherited from its parent (article)
      </p>
      <p class="custom">
        This element has style "font-size: 1.5em" so that the font-size will be 30px (20 * 1.5 = 30)
      </p>
    </article>
  </body>
</html>
```

style.css

```
article {
  font-size: 20px;
}

.custom {
  font-size: 1.5em;
}
```

We can check the computed styles of an element using the Developer Tools of the browser. First, go to the "**Elements**" tab, and select the <p class="custom"> element. Then, select the "**Computed**" panel (beside the "**Styles**" panel) and type "font-size" into the filter box. We can see that the font-size is 30px , as the screenshot points out.



Using the em unit can cause compounding errors like the below example.

Example 2:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="parent">
      I'm 15px
      <div class="child">
        I'm 30px, as expected
        <div class="child">I'm 60px, oh no!</div>
      </div>
    </div>
  </body>
</html>
```

style.css

```
.parent {
  font-size: 15px;
}

.child {
  font-size: 2em;
}
```

Therefore, instead of using em , we are recommended to use the rem unit.

1.3.2.2 rem

The root of an HTML document is the <html> element. If we do not define any font-size on the root element, 1rem equals the default browser font-size , 16px .

This unit is similar to em , but relative only to the root element, not any parent element. Thus, compounding does not occur as it does with the em units.

Example 1:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="parent">
   I'm 15px
   <div class="child">
    I'm 32px, as expected
    <div class="child">I'm 32px, good!</div>
   </div>
  </div>
 </body>
</html>
```

style.css

```
.parent {
 font-size: 15px;
}

.child {
 font-size: 2rem;
}
```

While using rem , it will be easier to calculate the font-size of elements if we set font-size of the <html> element to 10px .

If users change the browser's font-size , they will expect that font-size on the web page will change the same way. However, this will not happen if we set our root font-size to 10px . To avoid that, we will use a percentage of the default font-size of the browser like this:

```
html {
  /* 10px / 16px = 0.625 = 62.5% */
  font-size: 62.5%;
}
```

Example 2:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="parent">
      I'm 1.5rem = 1.5 * 10px = 15px
      <div class="child">
        I'm 2.6rem = 2.6 * 10px = 26px, as expected
        <div class="child">I'm 26px, good!</div>
      </div>
    </div>
  </body>
</html>
```

style.css

```
html {
  /* font-size: 10px; */
  font-size: 62.5%;
}

.parent {
  font-size: 1.5rem;
}

.child {
  font-size: 2.6rem;
}
```

In CSS media queries, rem and em do not depend on font-size of the HTML elements; instead, 1rem or 1em is always equal to 16px . Therefore, to convert px to em , we divide px by 16 and use the result in em unit. For example, 1000/16 = 62.5 , so 1000px equals 62.5em .

.4 Shorthand Properties

Some CSS declarations could have shorthand notation. For example, let's try the style of the element's border.

Example 1:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div></div>
  </body>
</html>
```

style.css

```
div {
  height: 50px;
  border-color: red;
  border-style: solid;
  border-top-width: 5px;
  border-right-width: 10px;
  border-bottom-width: 15px;
  border-left-width: 20px;
}
```

Shorthand notation of the above CSS is

```
div {
  height: 50px;
  border-color: red;
  border-style: solid;
  border-width: 5px 10px 15px 20px;
}
```

Example 2:

```
div {
  border-width: 5px 10px 15px;
```

```
}
```

equals

```
div {
  border-top-width: 5px;
  border-right-width: 10px;
  border-bottom-width: 15px;
  border-left-width: 10px; /* = border-right-width */
}
```

Example 3:

```
div {
  border-width: 5px 10px;
}
```

equals

```
div {
  border-top-width: 5px;
  border-right-width: 10px;
  border-bottom-width: 5px; /* = border-top-width */
  border-left-width: 10px; /* = border-right-width */
}
```

Example 4:

```
div {
  border-width: 5px;
}
```

equals

```
div {
  border-top-width: 5px;
  border-right-width: 5px;
  border-bottom-width: 5px;
  border-left-width: 5px;
}
```

.5 Where Do We Put Our CSS?

.5.1 Inline CSS

Inline CSS code is put in the HTML elements' style attribute. The styles are applied only to the elements that declare the " style " attribute.

In the below example, inline styles are applied only to the first paragraph since the second <p> element does not have any " style " attribute.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p style="color: red; font-size: 26px">This is the first paragraph.</p>
    <p>This is the second paragraph.</p>
  </body>
</html>
```

However, we should avoid inline CSS due to these reasons:

- it pollutes the HTML code
- it is hard to maintain both HTML and CSS code

For example, when we want to apply the color to all paragraphs, we must duplicate the CSS code in all <p> elements. After that, if we're going to change the color, we must change it in all those <p> elements.

.5.2 Internal CSS

Internal CSS is written as the content of the <style> element, and we often put it inside the <head> element.

Example:

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

```
    <title>My Website</title>
    <style type="text/css">
      p {
        color: red;
        font-size: 26px;
      }
    </style>
  </head>
  <body>
    <p>This is the first paragraph.</p>
    <p>This is the second paragraph.</p>
  </body>
</html>
```

It is not practical since we could have many lines of CSS code. In this case, that CSS code would pollute the HTML file and makes both CSS and HTML code hard to maintain.

..5.3 External CSS

External CSS is in an external CSS file (*.*css*) as below. Then, we need to tell the HTML file that we want to use that external CSS file by using the <link> element inside the <head> element. The <link> element has two attributes:

- rel : relationship of the linked document to the current document, so we set it to " stylesheet "
- href : this attribute specifies the URL of the linked resource

style.css

```
p {
  color: red;
  font-size: 28px;
}
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is the first paragraph.</p>
    <p>This is the second paragraph.</p>
  </body>
</html>
```

We can put the internal and external CSS anywhere inside the HTML file, but why should we put them inside the <head> element? Let's try this example to see the reason.

Example:

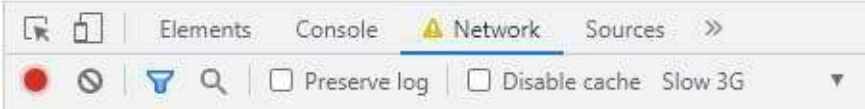
index.html

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

```
    <title>My Website</title>
  </head>
  <body>
    <p>This is the first paragraph.</p>
    <p>This is the second paragraph.</p>
    <link rel="stylesheet" href="style.css" />
  </body>
</html>
```

In the above example, we set style-sheet link as the last child of <body> . It means that the CSS file will be loaded after rendering all HTML above it.

Then, open the “Developer tools” of your browser (Chrome is used in this example) and set your **Network** to “**Slow 3G,**” as shown in the below screenshot.



Finally, refresh the “*index.html*” page to see that:

- Firstly, two paragraphs are loaded in black and small font size.
- Then, they are changed to big red texts when the CSS file is loaded.

It is not a good user experience since users see two styles of content and a flash or blink screen of style transition.

.6 User Agent Style Sheets

Besides your CSS code, there are the user agent’s style sheets. They are the default styles of web browsers, and different browsers could have different styles.

Example:

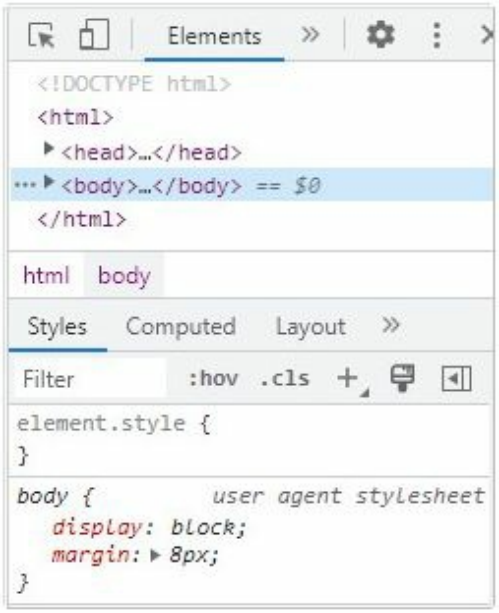
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <p>This is the first paragraph.</p>
  </body>
```

```
</html>
```

That simple HTML code without any CSS is applied default styles of the web browser as below.

```
body {
  display: block;
  margin: 8px;
}
```

To see it, open the Developer Tools of Google Chrome, then select the "**Elements**" tab; select the <body> element and switch to the "**Styles**" panel.



CHAPTER 12
CSS Selectors

.1 Universal Selector

Universal selector (“ * ”) is whatever element(s) but at least one element.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p>Some text</p>
    </div>
    <p>Another text</p>
  </body>
</html>
```

style.css

```
body * p {
  color: red;
}
```

Only “**Some text**” is red. “**Another text**” is a <p> element, but it is a direct child of the <body> element, so it was not selected.

.2 Type Selector

In CSS, we can select elements using HTML tag names such as “ div ,”“ p ,” etc. The syntax is:

```
tagname {
  declaration block
}
```

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p>Some text</p>
    </div>
    <p>Another text</p>
  </body>
</html>
```

style.css

```
p {
  color: red;
}
```

Result: All paragraphs are red.

.3 ID Selector

In CSS, we can select elements by using the “ id ” attribute of them. The syntax is:

```
#element-id {
  declaration block
}
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p id="my-text">Some text</p>
    </div>
    <p>Another text</p>
  </body>
</html>
```

style.css

```
#my-text {
  color: red;
}
```

Result: Only “**Some text**” is red.

.4 Class Selector

In CSS, we can select elements by using the “ class ” attribute of the elements. The syntax is:

```
.class-name {
  declaration block
}
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p class="my-text">Some text</p>
      <p class="my-text">Another text</p>
    </div>
    <p>Another text</p>
  </body>
</html>
```

style.css

```
.my-text {
  color: red;
}
```

Result: Only two paragraphs, children of the element, are red since they have the class “ my-text .”

.5 Multiple-class Selector

In CSS, we can select elements by using several classes (2 or more) in the “ class ” attribute of the elements. The syntax is:

```
.class-name-1.class-name-2...class-name-n {
  declaration block
}
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p class="my-text main">Some text</p>
      <p class="my-text">Another text</p>
    </div>
    <p>Another text</p>
  </body>
</html>
```

style.css

```
.my-text.main {
  color: red;
}

.my-text {
  color: blue;
}
```

Result: Only “**Some text**” is red since it has both “ my-text ” and “ main ” classes.

.6 CSS Pseudo-class

Pseudo-class is a CSS selector based on a specific state or position of HTML elements.

.6.1 Status of Links

.6.1.1 :link

In CSS, we use “ :link ” to select regular unvisited links.

 An <a> element without a“ href ” attribute is not a link.

.6.1.2 :visited

In CSS, we use “ :visited ” to select visited links. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <a>This is not a link</a>
    <a href="https://neodtruman.com" target="_blank">My Website</a>
    <a href="https://fonts.google.com" target="_blank">Google Fonts</a>
  </body>
</html>
```

style.css

```
:link {
  color: green;
}

:visited {
  color: red;
}
```

2.6.2 Cursor Interaction

2.6.2.1 :focus

In CSS, we use “ :focus ” to select elements having the focus.

2.6.2.2 :hover

In CSS, we use “ :hover ” to select elements with a mouse in front of them.

2.6.2.3 :active

In CSS, we use “ :active ” to select elements being clicked.

The order of these three pseudo-classes (focus > hover > active) in the CSS code is very important. It must be kept so that styles can be applied correctly.

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>Press TAB to remove the focus on the button.</p>
    <button>This is a button</button>
  </body>
</html>
```

style.css

```
:focus {
  color: red;
}

:hover {
  color: green;
```

```
}

:active {
  color: blue;
}
```

Click the button to see the effect.

We can add a tag name right before the Pseudo-class to select only the buttons as below.

```
button:focus {
  color: red;
}

button:hover {
  color: green;
}

button:active {
  color: blue;
}
```

2.6.3 Status of Elements

2.6.3.1 :enabled

In CSS, we use “ :enabled ” to select enabled elements.

2.6.3.2 :disabled

In CSS, we use “ :disabled ” to select disabled elements. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>Update user's information</p>
    <p>Username: <input type="text" value="username" disabled /></p>
    <p>Email: <input type="text" value="email@company.com" /></p>
  </body>
</html>
```

style.css

```
:enabled {
  background-color: cyan;
}

:disabled {
  background-color: #ccc;
}
```

2.6.3.3 :checked

In CSS, we use “ :checked ” to select checked radio buttons or checkboxes. For example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>Select your options:</p>
  <p>
   Gender:
   <label><input type="radio" name="gender" /> <span>Male</span></label>
   <label><input type="radio" name="gender" /> <span>Female</span></label>
  </p>
  <p>
   <label><input type="checkbox" /> <span>Accept the agreement</span></label>
  </p>
 </body>
</html>
```

style.css

```
input:checked + span {
 color: red;
}
```

2.6.4 Form Validation

2.6.4.1 :invalid

We use “ :invalid ” to select form elements if their values are not legal according to the element's settings. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <form action="">
      <p>This text input must be a three-letter code:</p>
      <input type="text" name="code" pattern="[A-Za-z]{3}" title="Three letter code." value="a" />
    </form>
  </body>
</html>
```

style.css

```
input:invalid {
  background-color: lightpink;
}
```

2.6.4.2 :required

We use “ :required ” to select form elements that are required. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <form action="">
```

```
    <p>This text input is required:</p>
    <input type="text" name="fname" required />
   </form>
  </body>
</html>
```

style.css

```
input:required {
  background-color: lightgreen;
}
```

2.6.4.3 *:out-of-range*

We use “ :out-of-range ” to select form elements with a value outside a specified range according to the min and max attributes. For example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <form action="">
      <p>This number input must be in the range of 1 - 6:</p>
      <input type="number" name="quantity" min="1" max="6" value="8" />
    </form>
  </body>
</html>
```

style.css

```
input:out-of-range {
  background-color: lightblue;
}
```

2.6.5 Child Elements

2.6.5.1 *:first-child*

In CSS, we use “ :first-child ” to select elements if they are the first child

of their parents.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div>
   <p>Line 1</p>
   <p>Line 2</p>
  </div>
  <p>Line 3</p>
  <p>Line 4</p>
 </body>
</html>
```

style.css

```
p:first-child {
 color: red;
}
```

Result: only “ Line 1 ” was selected because it is the first child.

2.6.5.2 :last-child

In CSS, we use “ :last-child ” to select elements if they are the last child of their parents.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div>
   <p>Line 1</p>
   <p>Line 2</p>
  </div>
```

```
    <p>Line 3</p>
    <p>Line 4</p>
  </body>
</html>
```

style.css

```
p:last-child {
  color: red;
}
```

Result: only “ Line 2 ” and “ Line 4 ” were selected because they are the last child.

2.6.5.3 :nth-child(N)

In CSS, we use “ :nth-child(N) ” to select elements if they are the N child of their parents.

Example 1:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p>Line 1</p>
      <p>Line 2</p>
      <p>Line 3</p>
    </div>
    <p>Line 4</p>
    <p>Line 5</p>
    <p>Line 6</p>
  </body>
</html>
```

style.css

```
p:nth-child(2) {
  color: red;
}
```

Result: only “ Line 2 ” and “ Line 4 ” were selected because they are the 2nd child.

:nth-child(2n+1), :nth-child(odd) will select odd-numbered children.

Example 2:

index.html

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

```
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p>Line 1</p>
      <p>Line 2</p>
      <p>Line 3</p>
    </div>
    <p>Line 4</p>
    <p>Line 5</p>
    <p>Line 6</p>
  </body>
</html>
```

style.css

```
p:nth-child(odd) {
  color: red;
}
```

Result: only “ Line 1 ,” “ Line 3 ” and “ Line 5 ” were selected because they are the odd child.

:nth-child(2n), :nth-child(even) will select even-numbered children.

Example 3:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p>Line 1</p>
      <p>Line 2</p>
      <p>Line 3</p>
    </div>
    <p>Line 4</p>
    <p>Line 5</p>
    <p>Line 6</p>
  </body>
```

```
</html>
```

style.css

```
p:nth-child(even) {
  color: red;
}
```

Result: only “ Line 2 ,” “ Line 4 ” and “ Line 6 ” were selected because they are the even child.

2.6.5.4 :nth-last-child(N)

:nth-last-child(N) is the reverted case of :nth-child(N) since it will examine the children from the bottom of the list.

2.6.5.5 :only-child

In CSS, we use “ :only-child ” to select elements if they are the only child of their parent.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div>
   <p>Line 1</p>
  </div>
  <p>Line 2</p>
  <p>Line 3</p>
 </body>
</html>
```

style.css

```
p:only-child {
 color: red;
}
```

Result: only “ Line 1 ” was selected because it is the only child.

2.6.6 Type of elements

2.6.6.1 :first-of-type

In CSS, we use “ :first-of-type ” to select the first child element of the specified element type.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div>
   <div>Line 1</div>
   <p>Line 2</p>
   <p>Line 3</p>
  </div>
  <p>Line 4</p>
 </body>
</html>
```

style.css

```
p:first-of-type {
 color: red;
}
```

Result: only “ Line 2 ” and “ Line 4 ” were selected because they are the first child.

2.6.6.2 *:last-of-type*

In CSS, we use “ :last-of-type ” to select the last child element of the specified element type.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>Line 1</p>
  <div>
   <p>Line 2</p>
   <p>Line 3</p>
   <div>Line 4</div>
  </div>
 </body>
</html>
```

style.css

```
p:last-of-type {
 color: red;
}
```

Result: only “ Line 1 ” and “ Line 3 ” were selected because they are the last child.

2.6.6.3 *:nth-of-type(N)*

In CSS, we use “ :nth-of-type(N) ” to select the Nth child element of the specified element type.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <div>Line 1</div>
      <p>Line 2</p>
      <p>Line 3</p>
    </div>
    <p>Line 4</p>
  </body>
</html>
```

style.css

```
p:nth-of-type(2) {
  color: red;
}
```

Result: only “ Line 3 ” was selected because it is the 2nd child.

2.6.6.4 *:nth-last-of-type(N)*

In CSS, we use " :nth-last-of-type(N) " to select the N^{th} last child element of the specified element type.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>Line 1</p>
    <div>
      <p>Line 2</p>
      <p>Line 3</p>
      <div>Line 4</div>
    </div>
  </body>
</html>
```

style.css

```
p:nth-last-of-type(2) {
  color: red;
}
```

Result: only " Line 2 " was selected because it is the 2^{nd} last child (counting from the bottom).

2.6.6.5 *:only-of-type*

In CSS, we use “ :only-of-type ” to select the only child element of the specified element type.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div>
   <div>Line 1</div>
   <p>Line 2</p>
   <p>Line 3</p>
  </div>
  <p>Line 4</p>
 </body>
</html>
```

style.css

```
p:only-of-type {
 color: red;
}
```

Result: only “ Line 4 ” was selected because it is the only child of type .

2.6.7 Others Pseudo-classes

2.6.7.1 :root

The “ :root ” pseudo-class selects the HTML document’s root element, the <html> element.

2.6.7.2 :empty

The “ :empty ” pseudo-class selects the empty elements.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div></div>
    <div></div>
    <div>This is not empty!</div>
  </body>
</html>
```

style.css

```
div:empty {
  height: 20px;
  border: 1px solid red;
}
```

2.6.7.3 *:target*

The " :target " pseudo-class selects the target element by its ID.

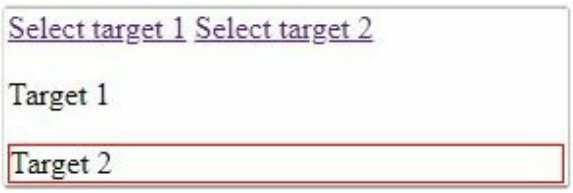
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <a href="#target1">Select target 1</a>
    <a href="#target2">Select target 2</a>
    <p id="target1">Target 1</p>
    <p id="target2">Target 2</p>
  </body>
</html>
```

style.css

```
:target {
  border: 1px solid red;
}
```

Result when the user clicks on the link "**Select target 2**":



2.6.7.4 *:not(selector)*

The " :not(selector) " pseudo-class is used to invert the selector .

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>Update user's information</p>
    <p>Address: <input type="text" value="Your address" /></p>
    <p>Email: <input type="text" value="email@company.com" /></p>
    <input type="submit" value="Submit" />
  </body>
</html>
```

style.css

```
input:not([type="submit"]) {
  background-color: cyan;
}
```

Result:

2.6.8 Pseudo-class Chain

We can chain pseudo-classes like the below example.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <a class="nav-lnk" href="#">Articles</a>
      <a class="nav-lnk" href="#">Links</a>
      <a class="nav-lnk" href="#">About</a>
      <a class="nav-lnk" href="#">Newsletter</a>
    </div>
  </body>
</html>
```

style.css

```
.nav-lnk:not(:first-child):not(:last-child) {
  color: red;
}
```

Result: Only the “**Links**” and “**About**” links are red since they are not the first or last child.

.7 CSS Pseudo-element

.7.1 ::first-letter

The “ ::first-letter ” pseudo-element selects the first letter of an element.

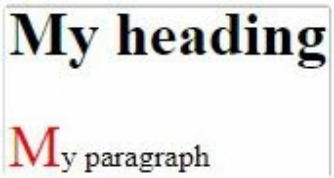
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>My heading</h1>
    <p>My paragraph</p>
  </body>
</html>
```

style.css

```
p::first-letter {
  font-size: 30px;
  color: red;
}
```

Result:



2.7.2 ::first-line

The “ ::first-line ” pseudo-element selects the first line of an element.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>My paragraph has<br />two lines.</p>
 </body>
</html>
```

style.css

```
p::first-line {
 color: red;
}
```

Result: Only “**My paragraph has**” is red.

2.7.3 ::before

The “ ::before ” pseudo-element generates and renders the content right before the element.

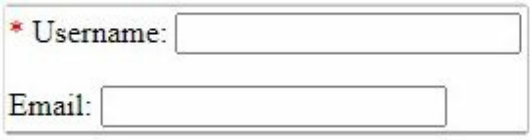
index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>
   <label class="required">Username: </label>
   <input type="text" />
  </p>
  <p>
   <label>Email: </label>
   <input type="text" />
  </p>
 </body>
</html>
```

style.css

```
.required::before {
 content: "* ";
 color: red;
}
```

Result:



2.7.4 ::after

The “ ::after ” pseudo-element generates and renders the content right after the element.

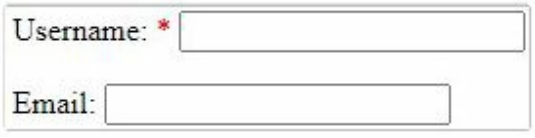
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>
      <label class="required">Username: </label>
      <input type="text" />
    </p>
    <p>
      <label>Email: </label>
      <input type="text" />
    </p>
  </body>
</html>
```

style.css

```
.required::after {
  content: "* ";
  color: red;
}
```

Result:



2.7.5 ::selection

The “ ::selection ” pseudo-element defines the style for the selected text.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>Please select some text in this paragraph.</p>
  </body>
</html>
```

style.css

```
p::selection {
  color: red;
  background-color: yellow;
}
```

.8 CSS Combined Selector

.8.1 Descendant selector

We can select all elements that are descendants of a specified element. The syntax is:

```
X Y {
  declaration blocks
}
```

The below example will select all children of elements having the class“ container .”

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div>
        <p>Line 1</p>
      </div>
      <p>Line 2</p>
    </div>
  </body>
</html>
```

style.css

```
.container p {
  color: red;
}
```

Result: Line 1 and Line 2 are red since they are descendants of the element.

2.8.2 Child Selector

We can select elements that are the immediate children of a specified element. The syntax is:

```
X > Y {
  declaration blocks
}
```

The below example will select all elements that are immediate children of elements having class " container ."

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div>
        <p>Line 1</p>
      </div>
      <p>Line 2</p>
    </div>
  </body>
</html>
```

style.css

```
.container > p {
  color: red;
}
```

Result: Only Line 2 is red since it is an immediate child of the element.

2.8.3 Adjacent Sibling Selector

We can select elements immediately following siblings of a specified element. The syntax is:

```
X + Y {
  declaration blocks
}
```

The below example will select all elements that follow a element (and are siblings).

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div>
        <p>Line 1</p>
      </div>
      <p>Line 2</p>
      <p>Line 3</p>
    </div>
  </body>
</html>
```

style.css

```
div + p {
  color: red;
}
```

Result: Only Line 2 is red.

2.8.4 General Sibling Selector

We can select elements that are siblings of a given element. The syntax is:

```
X ~ Y {
  declaration blocks
}
```

The example below will select all siblings of a element.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div>
        <p>Line 1</p>
      </div>
      <p>Line 2</p>
      <p>Line 3</p>
    </div>
  </body>
</html>
```

style.css

```
div ~ p {
  color: red;
}
```

Result: Line 2 and Line 3 are red since they are siblings of a element.

.9 CSS Attribute Selector

.9.1 [attribute]

We can select elements that have a specific attribute by using this syntax:

```
[attribute]
```

The example below will select <p> elements with the attribute “ class .”

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p class="my-text">Some text</p>
    </div>
    <p>Another text</p>
  </body>
</html>
```

style.css

```
[class] {
  color: red;
}
```

Result: Only “**Some text**” is red.

2.9.2 [attribute="value"]

We can select elements that have a specific attribute's value by using this syntax:

```
[attribute="value"]
```

The below example will select <p> elements having the attribute " class " with the value " my-text ."

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <p class="my-text">Some text</p>
    </div>
    <p class="another-text">Another text</p>
  </body>
</html>
```

style.css

```
[class="my-text"] {
  color: red;
}
```

Result: Only "**Some text**" is red.

2.9.3 [attribute|="value"]

We can select elements with an attribute that begins with a specific word (can be followed by a hyphen). The syntax is:

```
[attribute|="value"]
```

The below example will select elements that have the “ class ” attribute that begins with “ word ” (can be followed by a hyphen).

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>[class|="word"] { color: red; }</p>
  <p class="word">Word class: "word"</p>
  <p class="highlight word">Word class: "highlight word"</p>
  <p class="word highlight">Word class: "word highlight"</p>
  <p class="highlight-word">Word class: "highlight-word"</p>
  <p class="word-highlight">Word class: "word-highlight"</p>
  <p class="words">Word class: "words"</p>
 </body>
</html>
```

style.css

```
[class|="word"] {
  color: red;
}
```

Only these paragraphs are red:

- Word class: "word"
- Word class: "word-highlight"

2.9.4 [attribute~="value"]

We can select elements that have an attribute that contains a space-separated word. The syntax is:

```
[attribute~="value"]
```

The example below will select elements with the “ class ” attribute containing “ word .”

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>[class~="word"] { color: red; }</p>
  <p class="word">Word class: "word"</p>
  <p class="highlight word">Word class: "highlight word"</p>
  <p class="word highlight">Word class: "word highlight"</p>
  <p class="highlight-word">Word class: "highlight-word"</p>
  <p class="word-highlight">Word class: "word-highlight"</p>
  <p class="words">Word class: "words"</p>
 </body>
</html>
```

style.css

```
[class~="word"] {
  color: red;
}
```

Only these paragraphs are red:

- Word class: "word"
- Word class: "highlight word"
- Word class: "word highlight"

2.9.5 [attribute^="value"]

We can select elements with an attribute that starts with a specified value. The syntax is:

```
[attribute^="value"]
```

The example below will select elements with the “ class ” attribute that starts with “ word .”

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>[class^="word"] { color: red; }</p>
  <p class="word">Word class: "word"</p>
  <p class="highlight word">Word class: "highlight word"</p>
  <p class="word highlight">Word class: "word highlight"</p>
  <p class="highlight-word">Word class: "highlight-word"</p>
  <p class="word-highlight">Word class: "word-highlight"</p>
  <p class="words">Word class: "words"</p>
 </body>
</html>
```

style.css

```
[class^="word"] {
 color: red;
}
```

Only these paragraphs are red:

- Word class: "word"
- Word class: "word highlight"
- Word class: "word-highlight"
- Word class: "words"

.9.6 [attribute$="value"]

We can select elements with an attribute that ends with a specified value.

The syntax is:

```
[attribute$="value"]
```

The example below will select elements with the “ class ” attribute that ends with “ word .”

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>[class$="word"] { color: red; }</p>
  <p class="word">Word class: "word"</p>
  <p class="highlight word">Word class: "highlight word"</p>
  <p class="word highlight">Word class: "word highlight"</p>
  <p class="highlight-word">Word class: "highlight-word"</p>
  <p class="word-highlight">Word class: "word-highlight"</p>
  <p class="words">Word class: "words"</p>
 </body>
</html>
```

style.css

```
[class$="word"] {
 color: red;
}
```

Only these paragraphs are red:

- Word class: "word"
- Word class: "highlight word"
- Word class: "highlight-word"

2.9.7 [attribute*="value"]

We can select elements that have an attribute that contains a specified value. The syntax is:

```
[attribute*="value"]
```

The example below will select elements with the “ class ” attribute containing “ word .”

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p class="word">Word class: "word"</p>
  <p class="highlight word">Word class: "highlight word"</p>
  <p class="word highlight">Word class: "word highlight"</p>
  <p class="highlight-word">Word class: "highlight-word"</p>
  <p class="word-highlight">Word class: "word-highlight"</p>
  <p class="words">Word class: "words"</p>
 </body>
</html>
```

style.css

```
[class*="word"] {
 color: red;
}
```

Result: All paragraphs are red.

.10 Selector Grouping

We can group (using commas“ , ” as separators) many CSS selectors having the same declaration blocks into one to reduce the CSS file size.

For example, we can group these two selectors

```
div {
  color: red;
}

p {
  color: red;
}
```

into this

```
div,
p {
  color: red;
}
```

CHAPTER 13
CSS Cascading

When many duplicated CSS declarations apply to the same element, the web browser will pick up only one style and apply it to the page using the below cascading rules (the lower, the stronger).

.1 Rule 1: The Last Declaration Will Win

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
  <style>
   h1 {
    color: green;
   }
   h1 {
    color: blue;
   }
  </style>
 </head>
 <body>
  <h1>My first heading</h1>
  <p>My first paragraph.</p>
 </body>
</html>
```

style.css

```
h1 {
 color: yellow;
}
```

```
h1 {
  color: red;
}
```

Since we put the <link> element before the <style> element, all style declarations are in this order:

```
h1 { color: yellow; }
h1 { color: red; }
h1 { color: green; }
h1 { color: blue; }
```

Therefore, we will see "**My first heading**" in blue as the last declaration is " h1 { color: blue; } ."

If the external CSS (*style.css*) is put after the tag <style> , the style declarations are in this order:

```
h1 { color: green; }
h1 { color: blue; }
h1 { color: yellow; }
h1 { color: red; }
```

As a result, we will see "**My first heading**" in red.

.2 Rule 2: The More Specific Declaration Will Win

In this listing, the lower CSS selector is the more specific declaration:

- Universal Selector (*)
- Type Selector (e.g., h1, p, div , etc.)
- Combined Selector of Tags (e.g., “ body h1 ”)
- Class (.) or Pseudo-class(:) or Attribute([]) Selector
- Combined Selector of Classes and Tags (e.g., “ body .heading ”)
- Multiple-class Selector (e.g., “ .heading.first ”)
- ID (#) Selector

We will use this HTML file in the next six examples:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1 id="heading-one" class="heading first">My First Heading</h1>
    <p>My first paragraph.</p>
    <h1 id="heading-two" class="heading">My Second Heading</h1>
    <p>My second paragraph.</p>
  </body>
</html>
```

Example 1:

style.css

```
/* Type Selector */
h1 {
  color: orange;
}

/* Universal Selector */
* {
  color: red;
```

```
}
```

All headings are orange because “Type Selector” is more specific than “Universal Selector” despite the order.

Example 2:

style.css

```
/* Combined Selector of Tags */
body h1 {
  color: yellow;
}

/* Type Selector */
h1 {
  color: orange;
}
```

All headings are yellow because “Combined Selector of Tags” is more specific than “Type Selector” despite the order.

Example 3:

style.css

```
/* Class Selector */
.heading {
  color: green;
}

/* Combined Selector of Tags */
body h1 {
  color: yellow;
}
```

All headings are green because “Class Selector” is more specific than “Combined Selector of Tags” despite the order.

Example 4:

style.css

```
/* Combined Selector of Classes and Tags */
body .heading {
  color: blue;
}
```

```
/* Class Selector */
.heading {
  color: green;
}
```

All headings are blue because “Combined Selector of Classes and Tags” is more specific than “Class Selector” despite the order.

Example 5:

style.css

```
/* Multiple-Class Selector */
.heading.first {
  color: indigo;
}

/* Combined Selector of Classes and Tags */
body .heading {
  color: blue;
}
```

The first heading is indigo because the “Multiple-Class Selector” is more specific than the “Combined Selector of Classes and Tags” despite the order.

Example 6:

style.css

```
/* ID Selector */
#heading-one {
  color: violet;
}

/* Multiple-Class Selector */
.heading.first {
  color: indigo;
}
```

The first heading is violet because “ID Selector” is more specific than “Multiple-Class Selector” despite the order.

All-in-one example:

```
/* ID Selector */
#heading-one {
  color: violet;
}

/* Multiple-Class Selector */
.heading.first {
  color: indigo;
}

/* Combined Selector of Classes and Tags */
body .heading {
  color: blue;
}

/* Class Selector */
.heading {
  color: green;
}

/* Combined Selector of Tags */
body h1 {
  color: yellow;
}

/* Type Selector */
h1 {
  color: orange;
}

/* Universal Selector */
* {
  color: red;
}
```

Then open "**Chrome DevTools > Elements > Styles**" to check the result (as in the below screenshot) after selecting the first heading in the **Elements** panel.

The CSS declaration at the bottom of the CSS file will override the top. However, the “Universal Selector” at line 32 has the lowest priority, so the others override it.

.3 Rule 3: Inline Style Is Preferred to both Internal and External Styles

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <style>
      #heading-one {
        color: green;
      }
    </style>
  </head>
  <body>
    <h1 id="heading-one" style="color: red">My First Heading</h1>
    <p>My first paragraph.</p>
  </body>
</html>
```

Result: The heading is red.

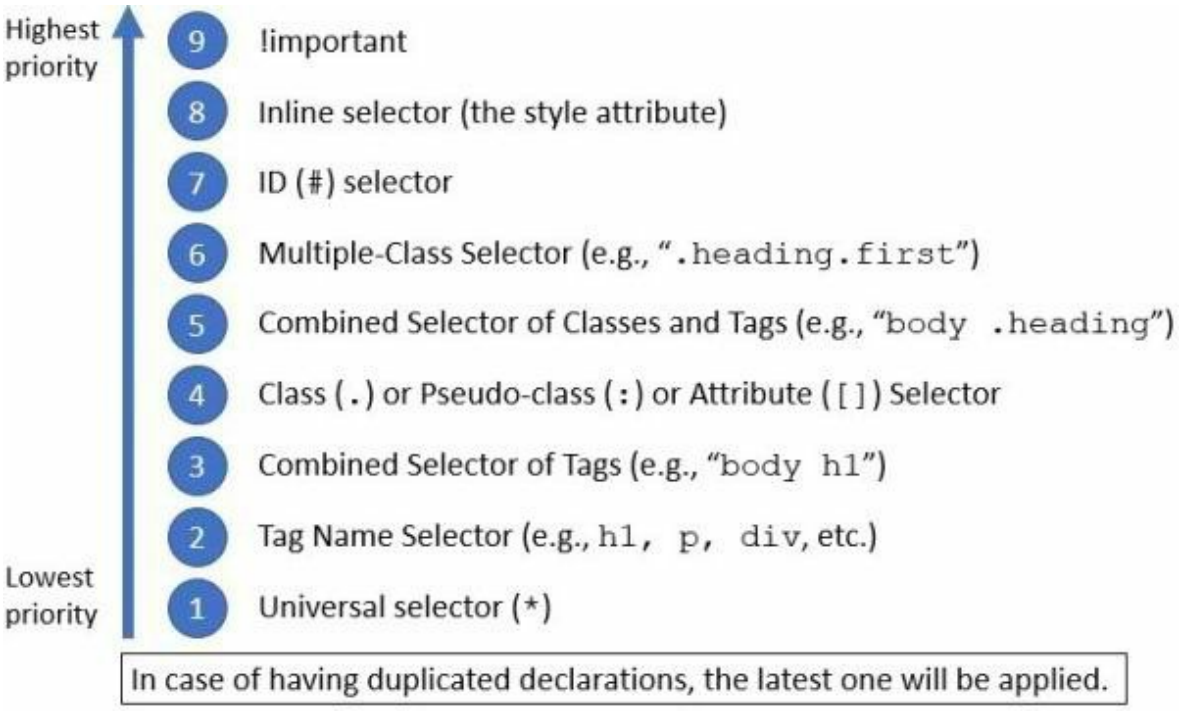
.4 Rule 4: “!important” Declaration Is the Strongest Rule

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <style>
   #heading-one {
    color: green !important;
   }
  </style>
 </head>
 <body>
  <h1 id="heading-one" style="color: red">My First Heading</h1>
  <p>My first paragraph.</p>
 </body>
</html>
```

Result: The heading is green.

Finally, this diagram summarizes the four rules above.



.5 Inheritance

Inheritance in CSS means that the parent's styles (*mostly related to text*) are applied to its children automatically. Many styles, such as the " border " property, are not inherited because that would be very impractical.

In some cases, as below, we must use the keyword " inherit " to set the element's style as the same as its parent. For example, we want to set the color of all children to green but keep the first one inherited from its parent (keep it in red).

index.html

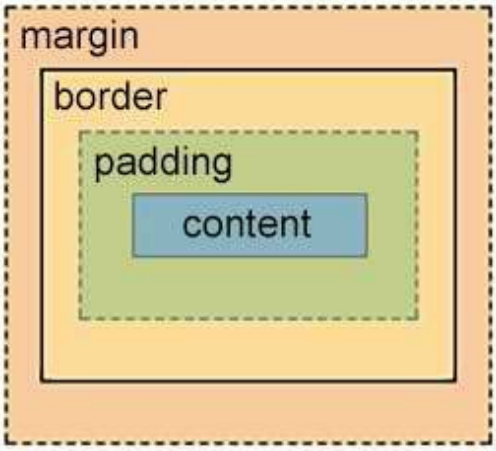
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <article class="parent">
      <h1>Heading</h2>
      <p class="line first-line">First line.</p>
      <p class="line second-line">Second line.</p>
    </article>
  </body>
</html>
```

style.css

```
.parent {
  color: red;
}

.line {
  color: green;
}

.first-line {
  color: inherit;
}
```

CHAPTER 14
CSS Box Model

An HTML element consists of a margin, border, padding, and content, as shown below.



To calculate the size of an element, we must consider its box-sizing type.

.1 Box Sizing Types

There are two box sizing types:

- border-box
- content-box

.1.1 Border Box

With " border-box " CSS, the padding and border are included in the total width and height of the element. Therefore, the actual size of the element is as CSS declarations(width: 200px; height: 200px in the below example).

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <pre id="my-element">
#my-element {
  width: 200px;
  height: 200px;
  padding: 10px;
  border: 5px solid red;
  margin: 8px;
  box-sizing: border-box;
}
    </pre>
    <span id="result"></span>
    <script>
      var myElement = document.getElementById("my-element");
      var width = myElement.offsetWidth;
      var height = myElement.offsetHeight;
      document.getElementById("result").innerHTML = "Element size: width = " + width + ", height
= " + height;
    </script>
  </body>
</html>
```

style.css

```
#my-element {
  width: 200px;
  height: 200px;
  padding: 10px;
  border: 5px solid red;
  margin: 8px;
  box-sizing: border-box;
}
```

.1.2 Content Box

With " content-box " CSS, the padding and border are not included in the total width and height of the element. Therefore, the actual size of the element must be added padding and border.

index.html: use the same code as in the "Border Box" section.

style.css

```
#my-element {
  width: 200px;
  height: 200px;
  padding: 10px;
  border: 5px solid red;
  margin: 8px;
  box-sizing: content-box;
}
```

Actual width = left-border + left-padding + css-width + right-padding + right-border

= 5 + 10 + 200 + 10 + 5 = 230

Actual height = top-border + top-padding + css-height + bottom-padding + bottom-border

= 5 + 10 + 200 + 10 + 5 = 230

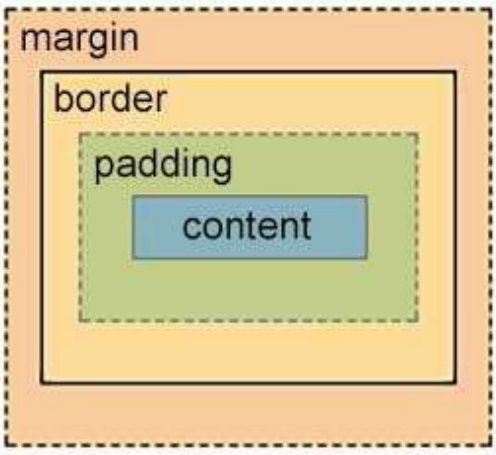
" box-sizing: content-box; " is the default CSS applied to all HTML elements in our web pages. However, this CSS rule confuses developers since the actual size is not as the " width " and " height " declarations. Therefore, it is a best practice to use this CSS rule to override the default box-sizing :

```
* {
  box-sizing: border-box;
}
```

.2 CSS Margins

The margin of an HTML element is the outer space from its border, as in

the below picture.



We have four CSS properties for the top, right, bottom, and left sides.

- margin-top
- margin-right
- margin-bottom
- margin-left

Inline elements do not have margin-top and margin-bottom properties.

And the shorthand for them is " margin ." For more details about the shorthand, please refer to the "Shorthand Properties" section.

Margin property can have one of the following values:

- auto - the browser calculates the margin automatically
- specific value in px, pt, cm , etc.
- % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element

Example:

index.html

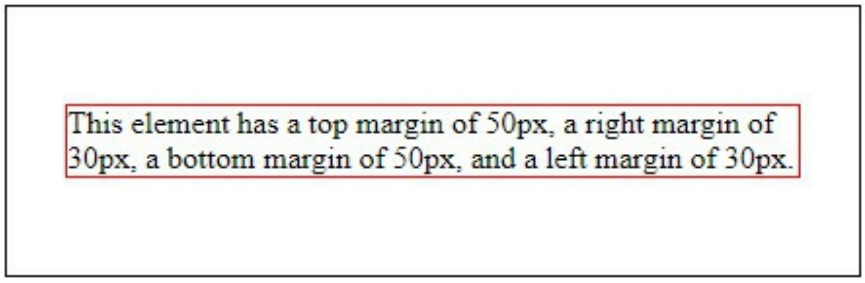
```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div id="parent">
      <div id="child">
        This element has a top margin of 50px, a right margin of 30px, a bottom margin of 50px, and a left margin of 30px.
      </div>
    </div>
  </body>
</html>
```

style.css

```
#parent {
  border: 1px solid black;
}

#child {
  border: 1px solid red;
  margin-top: 50px;
  margin-right: 30px;
  margin-bottom: 50px;
  margin-left: 30px;
}
```

Result:



.2.1 Auto Margin

We can horizontally align an element using “ margin: auto; ” as in the below example.

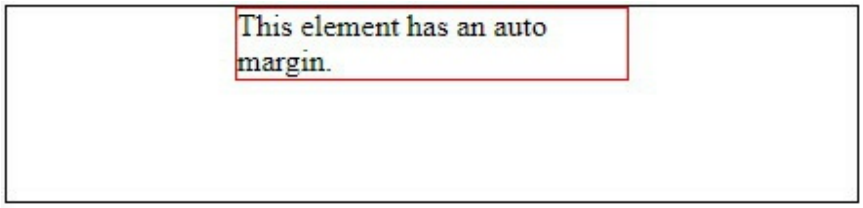
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div id="parent">
      <div id="child">This element has an auto margin.</div>
    </div>
  </body>
</html>
```

style.css

```
#parent {
  border: 1px solid black;
  height: 100px;
}

#child {
  border: 1px solid red;
  width: 200px;
  margin: auto;
}
```

Result:



.2.2 Collapsing Margins

The margin-bottom and margin-top could be collapsed, as in this example.

index.html

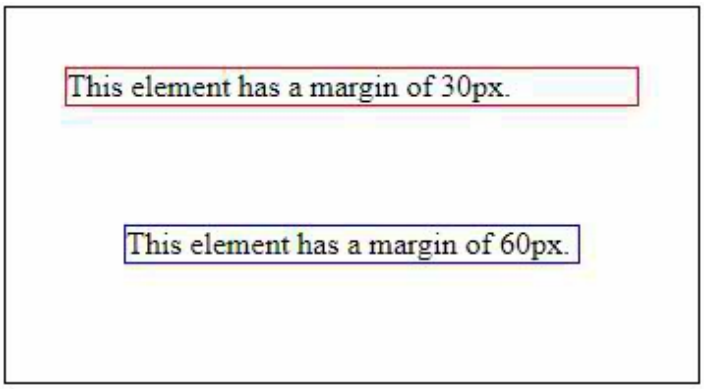
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
```

```
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div id="parent">
      <div id="child-one">This element has a margin of 30px.</div>
      <div id="child-two">This element has a margin of 60px.</div>
    </div>
  </body>
</html>
```

style.css

```
#parent {
  border: 1px solid black;
}

#child-one {
  border: 1px solid red;
  margin: 30px;
}

#child-two {
  border: 1px solid blue;
  margin: 60px;
}
```

The #child-one element has a bottom margin of 30px , and the #child-two element has a top margin of 60px , so the space (collapsing margin) between them will be 60px (the more significant number).



.3 CSS Outline

The outline is usually drawn outside the border but doesn't take up space like borders. These are outline properties:

```
outline-color, outline-style, outline-width
```

and the shorthand of them:

```
outline
```

Example:

```
p {
  outline-width: 5px;
  outline-style: dotted;
  outline-color: red;
}
```

will have a shorthand like this

```
p {
  outline: 5px dotted red;
}
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

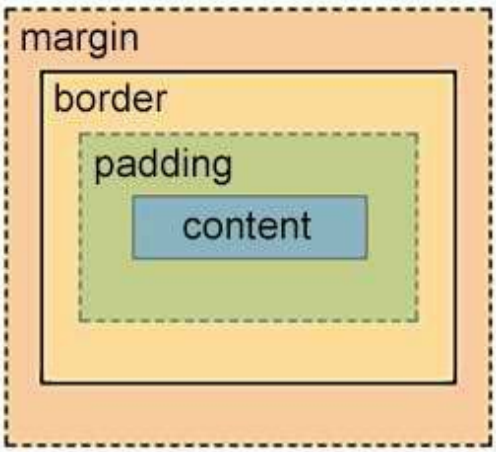
style.css

```
p {
  outline: 5px dotted red;
  border: 1px solid black;
}
```

The outline is drawn outside the border, as shown in this screenshot:

.4 CSS Borders

The border of an HTML element is the space between its padding and its margins, as in the below picture.



.4.1 Border Styles

We have CSS properties for the top, right, bottom, and left sides.

```
border-top-color, border-top-style, border-top-width
border-right-color, border-right-style, border-right-width
border-bottom-color, border-bottom-style, border-bottom-width
border-left-color, border-left-style, border-left–width
```

And shorthand of the above properties:

```
border-top, border-right, border-bottom, border-left
border-color, border- style, border-width
```

One shorthand for all properties:

```
border
```

For values of colors, please refer to the section “CSS Colors.”

Values of border style can be:

- dotted - defines a dotted border
- dashed - defines a dashed border
- solid - defines a solid border

- double - defines a double border
- groove - defines a 3D grooved border. The effect depends on the border-color value
- ridge - defines a 3D ridged border. The effect depends on the border-color value
- inset - defines a 3D inset border. The effect depends on the border-color value
- outset - defines a 3D outset border. The effect depends on the border-color value
- none - defines no border

Example 1:

```
p {
  border-top-color: red;
  border-top-style: solid;
  border-top-width: 5px;
}
```

will have a shorthand like this

```
p {
  border-top: 5px solid red;
}
```

Example 2:

```
p {
  border-top: 5px solid red;
  border-right: 5px solid red;
  border-bottom: 5px solid red;
  border-left: 5px solid red;
}
```

or

```
p {
  border-color: red;
  border-style: solid;
  border-width: 5px;
}
```

will have a shorthand like this

```
p {
  border: 5px solid red;
}
```

Example 3:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

style.css

```
p {
  border: 16px groove red;
}
```

4.2 CSS Border Radius

The “ border-radius ” property defines the radius of the HTML element's corners. The property value can be an absolute length or a relative length.

There are four properties for the corners:

```
border-top-left-radius
border-top-right-radius
border-bottom-right-radius
border-bottom-left-radius
```

A shorthand property for setting all four properties:

```
border-radius
```

Example:

index.html

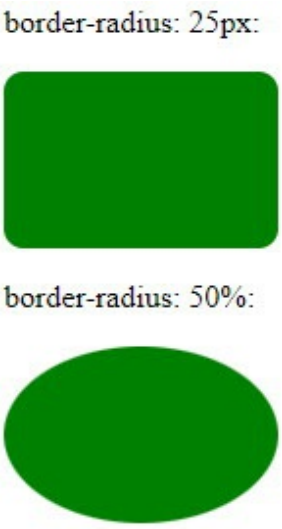
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>border-radius: 10px:</p>
    <p id="p1"></p>

    <p>border-radius: 50%:</p>
    <p id="p2"></p>
  </body>
</html>
```

style.css

```
p[id] {
  padding: 20px;
  width: 100px;
  height: 50px;
  background: green;
}

#p1 {
```

```
  border-radius: 10px;
}

#p2 {
  border-radius: 50%;
}
```

Result:



4.3 CSS Border Images

The “ border-image ” property allows you to specify an image to be used as a border.

The property has three parts:

- the image to use as the border
- the position to slice the image
- define whether the middle sections should be repeated or stretched

We will use the following image (called “*border.png*”):

The border-image property takes a PNG image and slices it into nine sections. It then places corners of the image at corners of the element, and the middle sections of the image are repeated or stretched as you specify.

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p id="border-img-round"></p>
    <p id="border-img-stretch"></p>
  </body>
</html>
```

style.css

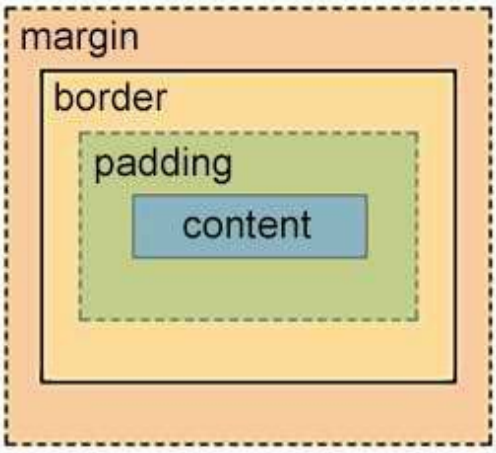
```
#border-img-round {
  border: 10px solid transparent;
```

```
  padding: 15px;
  border-image: url(border.png) 30 round;
}

#border-img-stretch {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 30 stretch;
}
```

Result:

.5 CSS Padding

Padding is the area between the element's content and its border.



The " padding " property is a shorthand property for the following individual padding properties:

```
padding-top, padding-right, padding-bottom, padding-left
```

Example 1:

```
p {
  padding-top: 20px;
  padding-right: 20px;
  padding-bottom: 20px;
  padding-left: 20px;
}
```

will have a shorthand like this

```
p {
  padding: 20px;
}
```

Example 2:

index.html

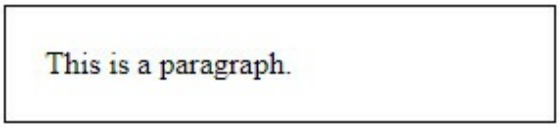
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
```

```
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

style.css

```
p {
  border: 1px solid black;
  padding: 20px;
}
```

Result:



.6 CSS Width and Height

We use “ width ” and “ height ” properties to set the content size of the elements. However, we must consider the box-sizing property to calculate the element's size.

We will use this HTML file in the next three examples:

index.html

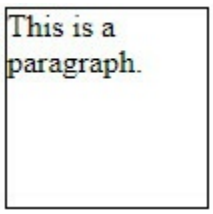
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

Example 1:

style.css

```
p {
  border: 1px solid black;
  width: 100px;
  height: 100px;
}
```

Result:



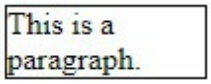
We use “ max-width ” and “ max-height ” properties to set the elements’ maximum content width and height.

Example 2:

style.css

```
p {
  border: 1px solid black;
  max-width: 100px;
  max-height: 100px;
}
```

Result:



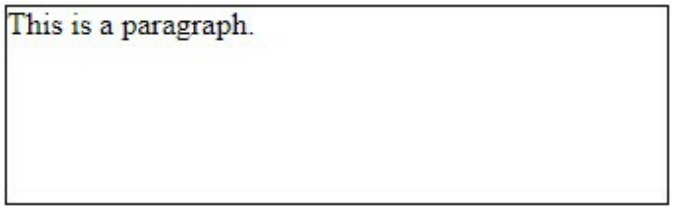
We use “ min-width ” and “ min-height ” properties to set the elements’ minimum content width and height.

Example 3:

style.css

```
p {
  border: 1px solid black;
  min-width: 100px;
  min-height: 100px;
}
```

Result:



CHAPTER 15
CSS Colors

We will use this HTML file in all the examples in this chapter:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

.1 Color Names

We can use color names to set the color of an element like this example. For example, the value can be: black, white, red, green, blue, yellow , etc.

Example:

style.css

```
p {
  background-color: red;
}
```

.2 HEX Colors

We can also set the color of an element using HEX value (0-9, A-F or a-f) with the below format.

```
#rrggbb or #rrggbbaa
```

or

```
#rgb or #rgba
```

where:

- r: red
- g: green
- b: blue
- a: alpha. This value sets how transparent the color is.

Example:

style.css

```
p {
  color: #ff0000;
  background-color: #ff0;
}
```

.3 RGB Colors

HTML elements can be set in color by using the RGB formula. The syntax is:

```
rgb(red, green, blue)
```

or

```
rgba(red, green, blue, alpha)
```

The red, green, and blue parameters are integers from 0 to 255 . The alpha parameter is a float number between 0 (fully transparent) and 1 (not transparent).

Example:

style.css

```
p {
  color: rgb(255, 0, 0);
  background-color: rgb(255, 255, 0);
}
```

.4 Grey Colors

When colors in all three channels (red, green, and blue) are the same, we get a grey color. For example:

```
#000, #111, #222, ..., #eee, #fff
```

or

```
rgb(0,0,0), rgb(17,17,17), rgb(34,34,34), ..., rgb(238,238,238), rgb(255,255,255)
```

CHAPTER 16
CSS Text

.1 Text Rendering

We will use this HTML file in the following examples:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph</p>
  </body>
</html>
```

.1.1 Setting Color

The “ color ” property is used to set the color of the text.

Example:

style.css

```
p {
  color: red;
}
```

.1.2 Text Decoration

The “ text-decoration-line ” property adds a decoration line to the text. Its value can be:

- underline - draw a line under the text

- line-through - draw a line through the text
- overline - draw a line over the text

Example:

style.css

```
p {
  text-decoration-line: line-through;
}
```

We can combine more than one value of the “ text-decoration-line ” property in a CSS declaration, like the following example.

The “ text-decoration-color ” property sets the color of the decoration line.

Example:

style.css

```
p {
  text-decoration-line: overline underline;
  text-decoration-color: red;
}
```

The “ text-decoration-thickness ” property sets the thickness of the decoration line.

Example:

style.css

```
p {
  text-decoration-line: underline;
  text-decoration-thickness: 5px;
}
```

The “ text-decoration-style ” property sets the style of the decoration line. Its value can be: solid (default value) , double, dotted, dashed, or wavy .

Example:

index.html

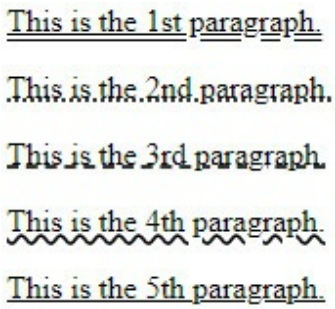
```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p class="p1">This is the 1st paragraph.</p>
    <p class="p2">This is the 2nd paragraph.</p>
    <p class="p3">This is the 3rd paragraph.</p>
    <p class="p4">This is the 4th paragraph.</p>
    <p>This is the 5th paragraph.</p>
  </body>
</html>
```

style.css

```
p {
  text-decoration-line: underline;
}

.p1 {
  text-decoration-style: double;
}

.p2 {
  text-decoration-style: dotted;
}

.p3 {
  text-decoration-style: dashed;
}

.p4 {
  text-decoration-style: wavy;
}
```

Result:



The " text-decoration " property is a shorthand property for the four properties above.

Example:

style.css

```
p {
  text-decoration: underline red wavy 3px;
}
```

5.1.3 Text Transform

The " text-transform " property is used to turn everything into upper-case or lower-case letters or capitalize the first letter of each word. Its value can be uppercase, lowercase , or capitalize .

Example:

style.css

```
p {
  text-transform: capitalize;
}
```

5.1.4 Handling Whitespaces

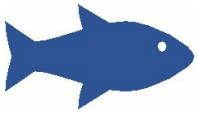
The " white-space " property handles whitespaces inside an HTML element. Typically, browsers will collapse a sequence of whitespace characters within an element into a single space character.

The white-space property can have one of these values:

- normal (default value) - This will collapse all sequences of whitespace into a single space character. It also breaks the line if it

exceeds the containing element's width.

- nowrap : ensures that sequences of whitespace or line breaks will collapse into a single space character. It usually causes a horizontal scroll bar since there is no line break.
- pre : ensures that sequences of whitespace won't collapse. Lines are only broken at new lines, as in the HTML code.
- pre-line : likes pre , but line-breaks will happen if lines exceed the width of containing element. The sequences of whitespaces in a line will be collapsed.
- pre-wrap : likes pre , but line-breaks will happen if lines exceed the width of containing element. The sequences of whitespaces in a line won't be collapsed.



Please disable the "Prettier - Code formatter" extension before trying the below example.

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>This is the heading</h1>
    <p>
      This is some text. This is some text. This is some text.      This is some text. This is some text.
This is some text.
      This is some text. This is some text. This is some text.
      This is some text. This is some text. This is some text.
    </p>
  </body>
</html>
```

style.css

```
p {
  white-space: pre-wrap;
}
```

5.1.5 Text Overflow

The “ text-overflow ” property specifies how the overflowed content that its container will hide should be signaled to the visitors.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p class="test-clip">text-overflow: clip - This is some text</p>
    <p class="test-ellipsis">text-overflow: ellipsis - This is some text</p>
  </body>
</html>
```

style.css

```
.test-clip {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: clip;
}

.test-ellipsis {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

Result:

text-overflow: clip - This is son

text-overflow: ellipsis - This…

5.1.6 Word Wrap

The “ word-wrap ” property allows long words to be broken and wrapped onto the next line by setting its value to “ break-word .”

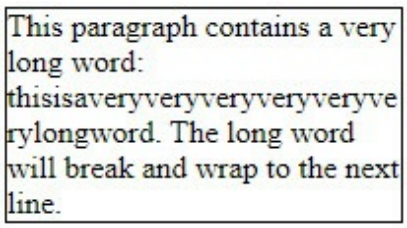
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p class="test-word-wrap">
      This paragraph contains a very long word:
      thisisaveryveryveryveryveryverylongword. The long word will break and wrap to the next line.
    </p>
  </body>
</html>
```

style.css

```
.test-word-wrap {
  width: 200px;
  border: 1px solid #000000;
  word-wrap: break-word;
}
```

Result:



5.1.7 Word Break

The “ word-break ” property specifies line-breaking rules. This property can have one of these values:

- keep-all : this will break the whole word into a new line or break at a hyphen
- break-word : this will break long words at arbitrary points into a new line or break at a hyphen to prevent overflow
- break-all : the lines will break at any character

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>Testing "word-break: keep-all"</p>
    <p class="test-keep-all">This is a-very-long-hyphen word, and this is
averyveryveryveryveryverylong word.</p>

    <p>Testing "word-break: break-word"</p>
    <p class="test-break-word">This is a-very-long-hyphen word, and this is
averyveryveryveryveryverylong word.</p>

    <p>Testing "word-break: break-all"</p>
    <p class="test-break-all">This is a-very-long-hyphen word, and this is
averyveryveryveryveryverylong word.</p>
  </body>
</html>
```

style.css

```
p[class] {
  width: 140px;
  border: 1px solid #000000;
}

.test-keep-all {
  word-break: keep-all;
```

```
}

.test-break-word {
  word-break: break-word;
}

.test-break-all {
  word-break: break-all;
}
```

Result:

Testing "word-break: keep-all"

This is a-very-long-hyphen word, and this is averyveryveryveryveryverylong word.

Testing "word-break: break-word"

This is a-very-long-hyphen word, and this is averyveryveryveryveryverylong word.

Testing "word-break: break-all"

This is a-very-long-hyphen word, and this is averyveryveryveryveryverylong word.

.2 Text Spacing

We will use this HTML file in all the examples in this section:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This paragraph is for testing.</p>
    <p class="test-one">This paragraph is for testing.</p>
    <p class="test-two">This paragraph is for testing.</p>
  </body>
</html>
```

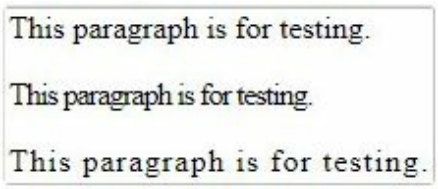
.2.1 Letter Spacing

The “ letter-spacing ” property is used to specify the space between the characters in a text.

style.css

```
.test-one {
  letter-spacing: -1px;
}

.test-two {
  letter-spacing: 1px;
}
```

Result:



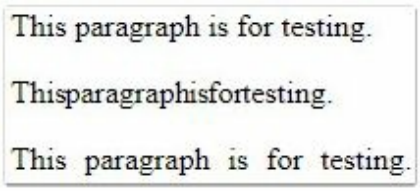
.2.2 Word Spacing

The “ word-spacing ” property is used to specify the space between the words in a text.

style.css

```
.test-one {
  word-spacing: -5px;
}

.test-two {
  word-spacing: 5px;
}
```

Result:



5.2.3 Line Height

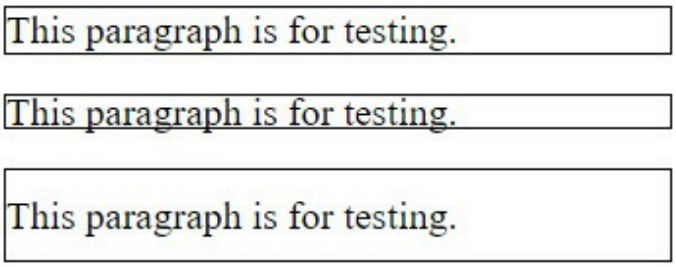
The “ line-height ” property is used to specify the space between lines. Its value can be:

- a number that will be multiplied by the current font-size to set the line height
- %: a line height in percent of the current font size
- a fixed line height (in px, pt, cm , etc.).

style.css

```
p {
  border: 1px solid black;
  font-size: 20px;
}

.test-one {
  line-height: 80%; /* 20px * 80% = 16px */
}

.test-two {
  line-height: 2.3; /* 20px * 2.3 = 46px */
}
```

Result:



The “ line-height ” property can vertically align an element inside its containing element by setting its value to the equal height of the containing element.

index.html

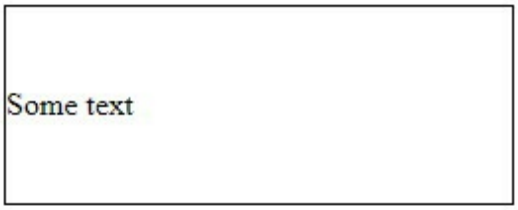
```
<!DOCTYPE html>
<html>
```

```
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <span>Some text<span>
    </div>
  </body>
</html>
```

style.css

```
div {
  height: 100px;
  border: 1px solid black;
}

span {
  line-height: 100px;
}
```

Result:



.3 Text Position

.3.1 Text Align

The “text-align” property is used to set the horizontal alignment of a text. Its value can be left, right, center , or justify .

index.html

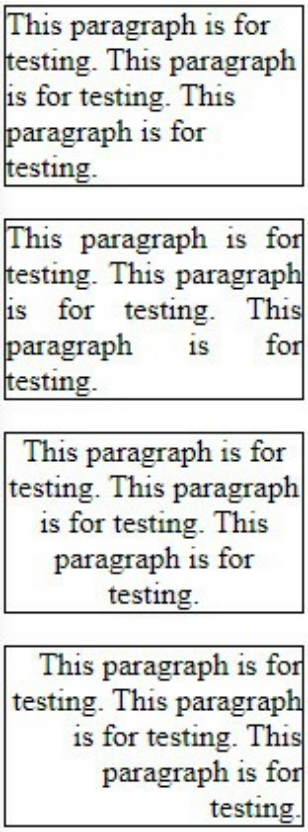
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.</p>
    <p class="test-one">This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.</p>
    <p class="test-two">This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.</p>
    <p class="test-three">This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.</p>
  </body>
</html>
```

style.css

```
p {
  width: 150px;
  border: 1px solid black;
}

.test-one {
  text-align: justify;
}

.test-two {
  text-align: center;
}

.test-three {
  text-align: right;
```

```
}
```

Result:


This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.
This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.
This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.
This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.

5.3.2 Text Indent

The “ text-indent ” property is used to specify the indentation of the first line of a text.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <p>
   This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.
  </p>
  <p class="test">
   This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.
  </p>
 </body>
</html>
```

style.css

```
p {
 width: 150px;
 border: 1px solid black;
}

.test {
 text-indent: 30px;
}
```

Result:

This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.

This paragraph is for testing. This paragraph is for testing. This paragraph is for testing.

.4 CSS Fonts

We will use this HTML file in all the examples in this section:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This Paragraph Is for Testing.</p>
  </body>
</html>
```

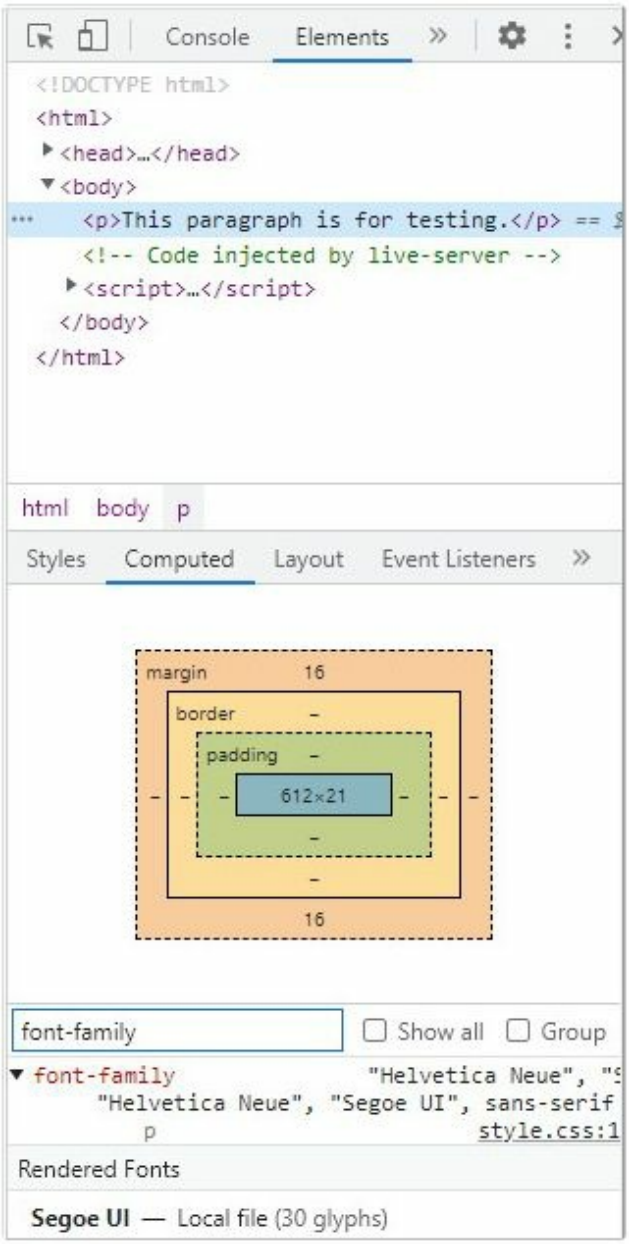
.4.1 Font Family

As a fallback system, the “ font-family ” property should hold several font names (separated by commas). If the browser does not support the first font, it will try the next one.

style.css

```
p {
  font-family: "Helvetica Neue", "Segoe UI", sans-serif;
}
```

We can check the rendered font of an element using the Developer Tools of the browser. First, go to the “**Elements**” tab, and select the <p> element. Then, select the “**Computed**” panel (beside the “**Styles**” panel). We can see at the bottom of the screenshot that the rendered font is “ Segoe UI .”



I used Chrome to test the example. Since my Chrome does not have the font “ Helvetica Neue ,” it uses “ Segoe UI ” to render the text.

5.4.2 Font Size

The “ font-size ” property sets the size of the text.

style.css

```
html {
  /* font-size: 10px; */
  font-size: 62.5%;
}

p {
  font-size: 2.5rem;
}
```

We usually use rem as the unit of font-size since it helps us to scale elements on our web pages by changing only the root font-size .

Please review the “CSS Units > Relative Lengths > rem” section for more examples.

5.4.3 Font Weight

The “ font-weight ” property specifies the weight of a font. Font weight can be bold, normal , etc., or a number from thin to thick characters (100, 200, 300, ..., 900).

style.css

```
p {
  font-weight: bold;
}
```

5.4.4 Font Style

The “ font-style ” property is mainly used to specify italic text. This property has one of these values:

- italic - The text is shown in italics
- normal - The text is shown normally

style.css

```
p {
  font-style: italic;
}
```

5.4.5 Font Variant

The “ font-variant ” property specifies whether or not a text should be displayed in a “ small-caps ” font. In small caps, all lowercase letters are converted to uppercase, but the converted uppercase letters will appear in smaller font sizes than the original uppercase letters.

style.css

```
p {
  font-variant: small-caps;
}
```

Result: The “F” character is small-cap since it was not an uppercase letter at the beginning.

5.4.6 The @font-face

This special CSS is used to define your font. Most browsers support these font types:

- TrueType Fonts (*.ttf)
- OpenType Fonts (*.otf)
- The Web Open Font Format (*.woff)

You can download fonts here:

https://fonts.google.com

since Google Fonts is a library of 1000+ open-source font families for your website.

style.css

```
@font-face {
  font-family: "Dyna Puff";
  src: url(DynaPuff-Regular.ttf);
}

p {
  font-family: "Dyna Puff";
}
```

Result:

CHAPTER 17
CSS Backgrounds

.1 CSS Background Properties

The CSS background properties are used to add backgrounds for HTML elements.

We will use this HTML file in all the examples in this section:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

'.1.1 background-color

The “ background-color ” property sets the background color of elements.

style.css

```
p {
  background-color: yellow;
}
```

Result:

This is a paragraph.

'.1.2 background-image

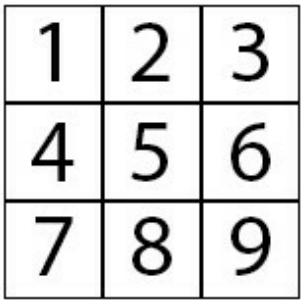
The “ background-image ” property uses one or more images to render the background of elements.

Example 1:

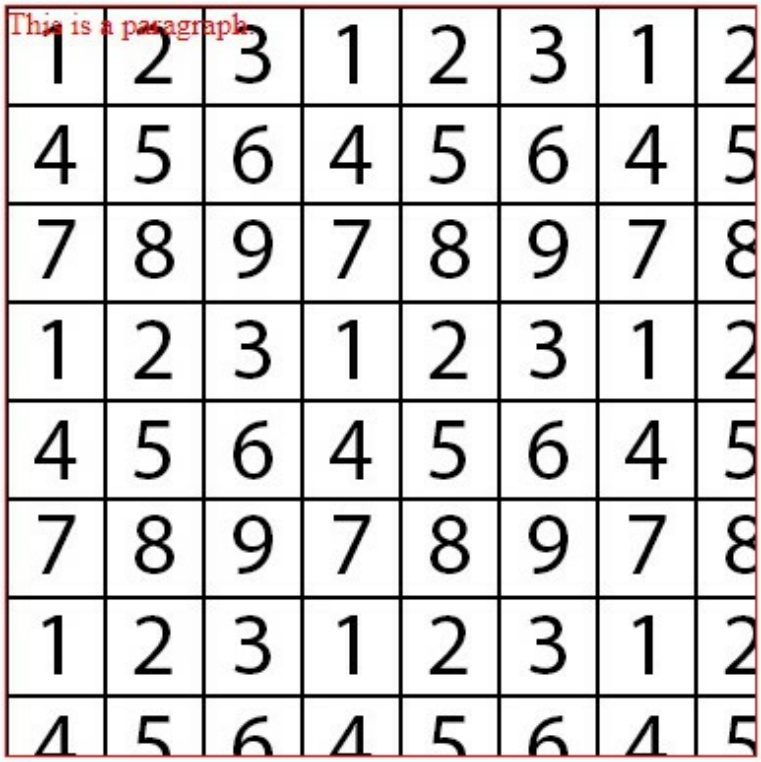
style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 380px;
  height: 380px;
  background-image: url("bg.jpg");
}
```

In the example, I use the “*bg.jpg*” file, which is an image of 150x150 pixels, and here it is:



Result:



It can be seen that the last images at the right and bottom edges have been clipped as they do not fit into the element.

We can add multiple images to an element's background like this.

Example 2:

```
p {
  color: red;
  border: 1px solid red;
  width: 380px;
  height: 380px;
  background-image: url("another-bg.png"), url("bg.jpg");
}
```

In example 2, the first image is placed on top of the second. Therefore, the first must have a transparent background (using a PNG file) so the users can see both images.

We can see that the images are repeated horizontally and vertically. To control the repeating, we use the " background-repeat " property.

'.1.3 background-repeat

The “ background-repeat ” property specifies how the background image is repeated. This property can have one of these values:

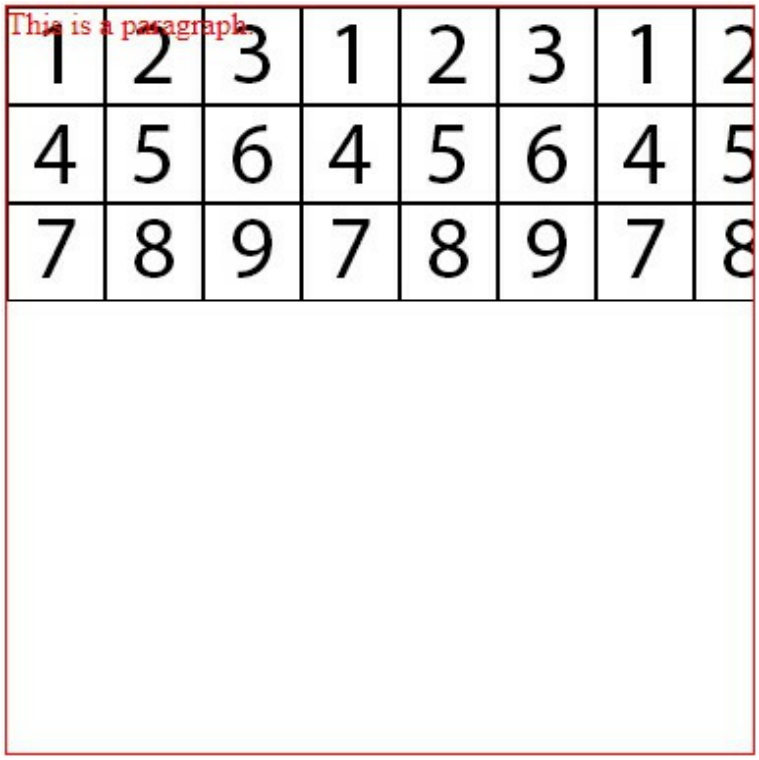
- repeat (default value) - The background image is repeated vertically and horizontally.
- repeat-x - The background image is repeated only horizontally.
- repeat-y - The background image is repeated only vertically.
- space - The background image is repeated as much as possible without clipping, and whitespace is distributed evenly between the images.
- round - The background image is repeated and scaled to fill all the space without clipping or gaps.
- no-repeat - The background image will only be shown once.

Example 1:

style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 380px;
  height: 380px;
  background-image: url("bg.jpg");
  background-repeat: repeat-x;
}
```

Result:

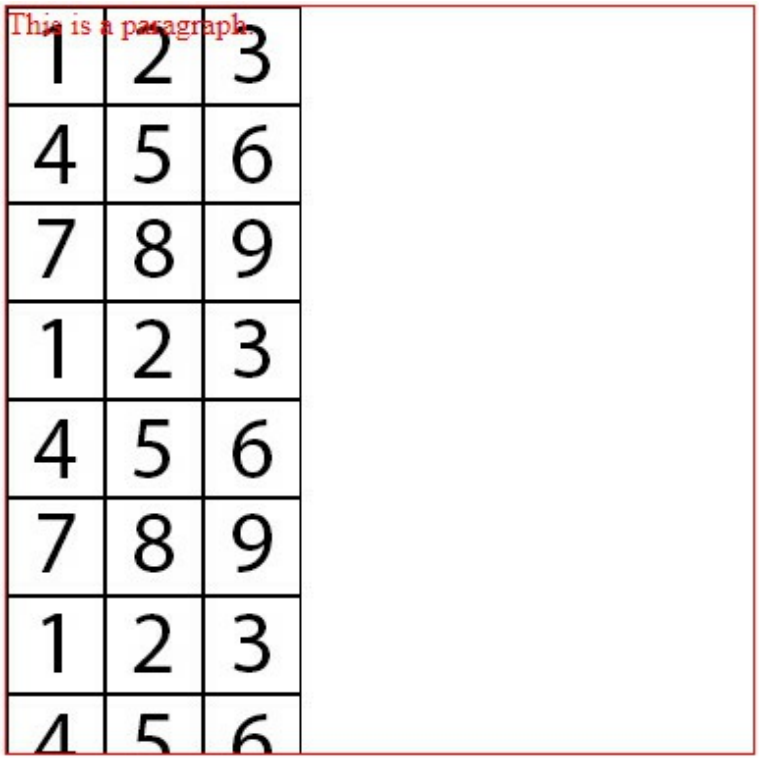


Example 2:

style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 380px;
  height: 380px;
  background-image: url("bg.jpg");
  background-repeat: repeat-y;
}
```

Result:

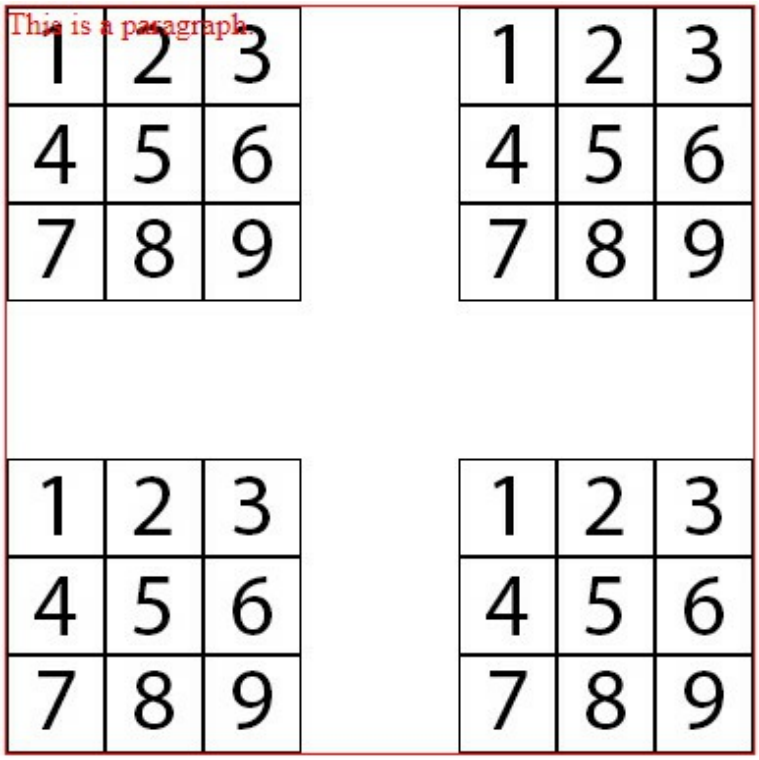


Example 3:

style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 380px;
  height: 380px;
  background-image: url("bg.jpg");
  background-repeat: space;
}
```

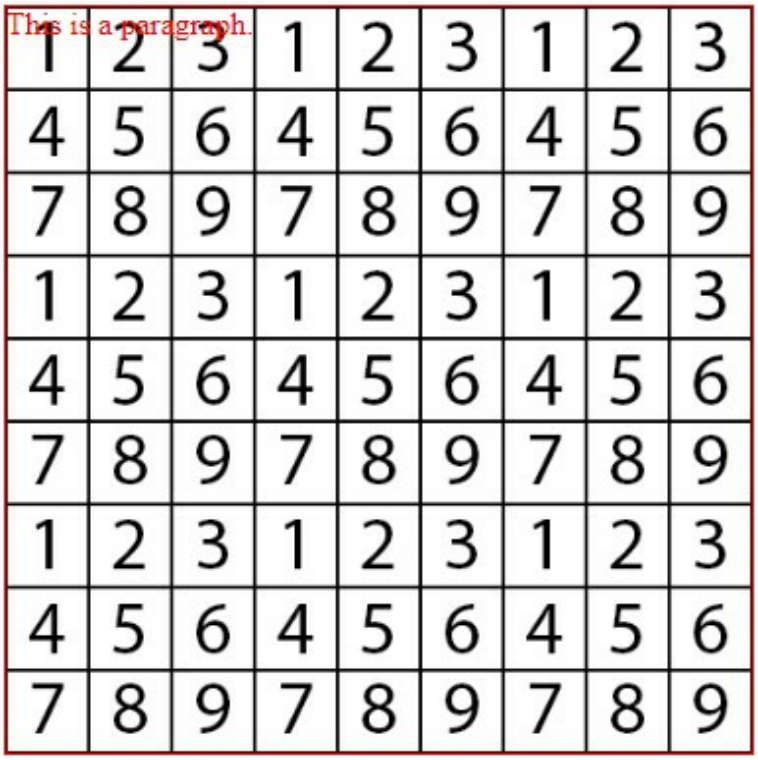
Result:



Example 4:

style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 380px;
  height: 380px;
  background-image: url("bg.jpg");
  background-repeat: round;
}
```



Example 5: we can set the property to “ no-repeat ” to stop the repeating.

```
p {
  color: red;
  border: 1px solid red;
  width: 380px;
  height: 380px;
  background-image: url("bg.jpg");
  background-repeat: no-repeat;
}
```

'.1.4 background-position

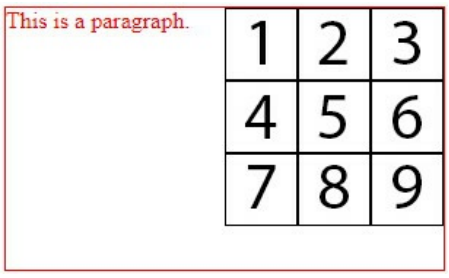
The “ background-position ” property specifies the starting position of the background image. This property can have one of these values:

- x y - where the top left corner is “0 0.”
- x% y% - The top left corner is “0% 0%” and the right bottom corner is “100% 100%.”
- or one of these: left top, left center, left bottom, right top, right center, right bottom, center top, center center, center bottom .

style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 300px;
  height: 180px;
  background-image: url("bg.jpg");
  background-repeat: no-repeat;
  background-position: right top;
}
```

Result:



'.1.5 background-size

The “ background-size ” property allows you to change the size of the background image.

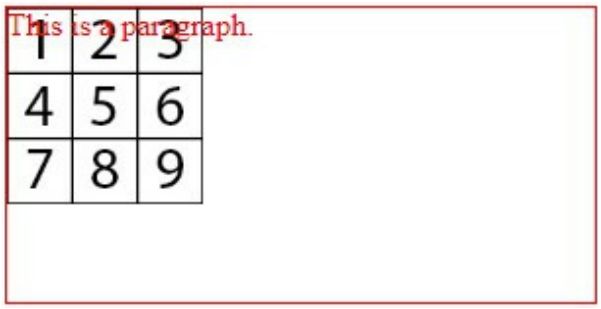
- w h - width and height of the background image
- w% h% - width and height of the background image in percent of the containing element.
- contain - scales the background image to be as large as possible but keeps it inside the bounding box of the containing element.
- cover - scales the background image to be as large as possible so that it will cover all its parent content.

Example 1:

style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 300px;
  height: 150px;
  background-image: url("bg.jpg");
  background-repeat: no-repeat;
  background-size: 100px 100px;
}
```

Result:



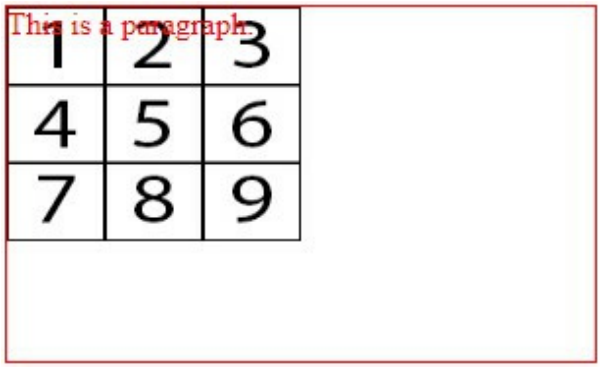
Example 2:

style.css

```
p {
```

```
  color: red;
  border: 1px solid red;
  width: 300px;
  height: 180px;
  background-image: url("bg.jpg");
  background-repeat: no-repeat;
  background-size: 50% 66%;
}
```

Result:

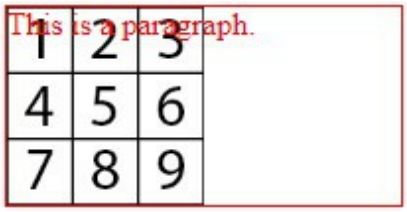


Example 3:

style.css

```
p {
  color: red;
  border: 1px solid red;
  width: 200px;
  height: 100px;
  background-image: url("bg.jpg");
  background-repeat: no-repeat;
  background-size: contain;
}
```

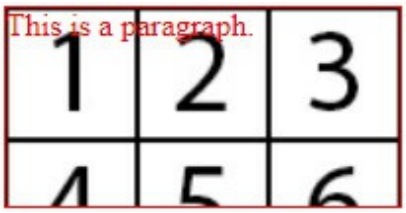
Result: The “contain” keyword scales the background image to be as large as possible but keeps the image inside the bounding box containing the element.



Example 4:

```
p {
  color: red;
  border: 1px solid red;
  width: 200px;
  height: 100px;
  background-image: url("bg.jpg");
  background-repeat: no-repeat;
  background-size: cover;
}
```

Result: The “ cover ” keyword scales the background image to be as large as possible, and the image must cover all the content of the containing element.



'.1.6 background

We can use the shorthand of background properties like the below example. Please use a background image smaller than 300x300 pixels in the below example.

style.css

```
p {
  width: 300px;
  height: 300px;
  background: #ff0 url("bg.jpg") no-repeat right top;
}
```

'.1.7 background-clip

The “ background-clip ” property specifies the drawing area of the background (image or color). The property can have one of these values:

- border-box - (default) the background is painted to the outside edge of the border
- padding-box - the background is painted to the outside edge of the padding
- content-box - the background is painted within the content box

Example:

index.html

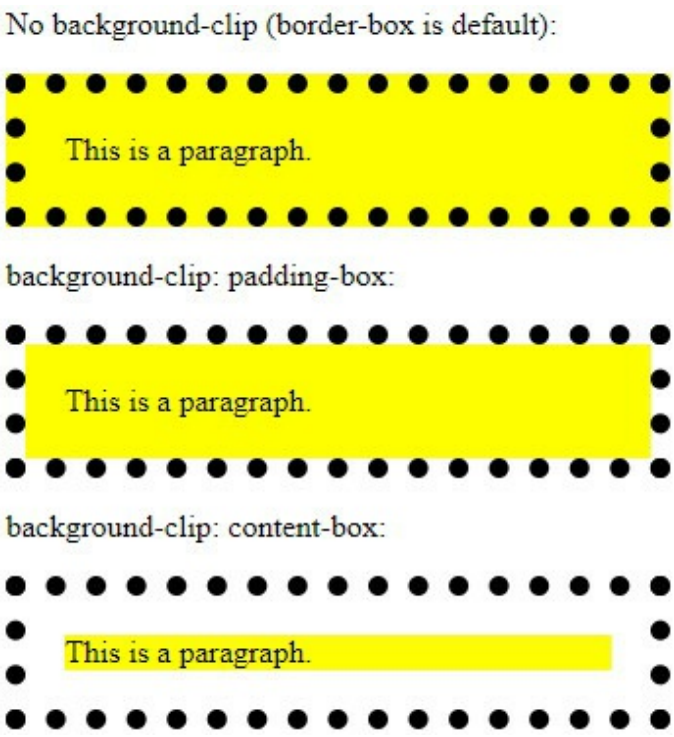
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>No background-clip (border-box is default):</p>
    <p class="bg">This is a paragraph.</p>

    <p>background-clip: padding-box:</p>
    <p class="bg padding">This is a paragraph.</p>
```

```
    <p>background-clip: content-box:</p>
    <p class="bg content">This is a paragraph.</p>
  </body>
</html>
```

style.css

```
.bg {
  border: 10px dotted black;
  padding: 20px;
  background: yellow;
}

.padding {
  background-clip: padding-box;
}

.content {
  background-clip: content-box;
}
```

Result:



'.1.8 background-origin

The “ background-origin ” property specifies where the background image is positioned. The property can have one of these values:

- border-box - the background image starts from the border
- padding-box - (default) the background image starts from the padding edge
- content-box - the background image starts from the edge of the content

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>No background-origin (padding-box is default):</p>
    <p class="bg">Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>

    <p>background-origin: border-box:</p>
    <p class="bg border-box">Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>

    <p>background-origin: content-box:</p>
    <p class="bg content-box">Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
  </body>
</html>
```

style.css

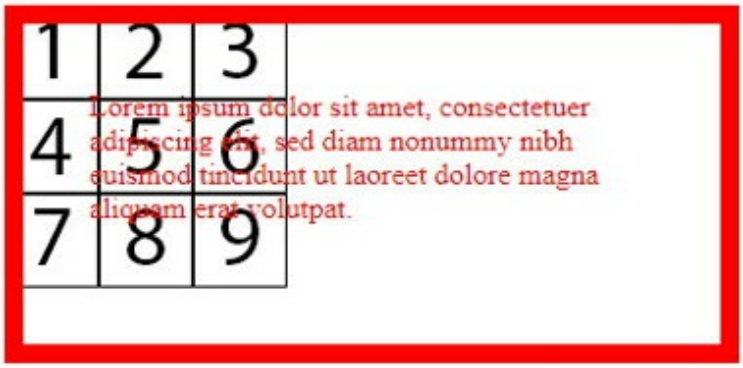
```
.bg {
  width: 300px;
  height: 100px;
  color: red;
  border: 10px solid red;
```

```
  padding: 35px;
  background: url(bg.jpg);
  background-repeat: no-repeat;
}

.border-box {
  background-origin: border-box;
}

.content-box {
  background-origin: content-box;
}
```

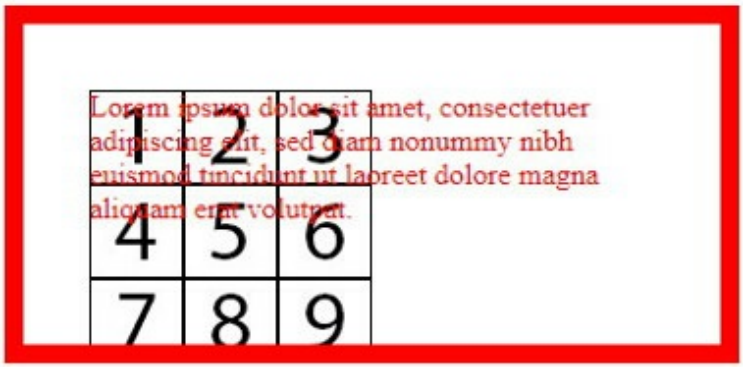
Result:

No background-origin (padding-box is default):

background-origin: border-box:



background-origin: content-box:



'.1.9 background-attachment

The “ background-attachment ” property specifies if the background image scrolls with the rest of the page or not. This property can have one of these values:

- scroll (default value) - The background image will scroll with the page.
- fixed - The background image will not scroll with the page.
- local - The background image will scroll with the element's contents.

Example 1:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <header>This is a header with height of 60px.</header>
  <div class="fixed"></div>
  <footer></footer>
 </body>
</html>
```

style.css

```
body {
 margin: 0;
}

header {
 height: 60px;
}

.fixed {
 background-position: center 60px;
 height: 500px;
 background-image: url("bg.jpg");
 background-repeat: no-repeat;
```

```
  background-attachment: fixed;
}

footer {
  height: 1000px;
  background-color: lightgreen;
}
```

Example 2: The background image can be scrolled with the element's content by using the " background-attachment " property and setting it as " local ."

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
  </body>
</html>
```

style.css

```
p {
  width: 120px;
  height: 200px;
  overflow: auto;
  border: 1px solid black;
  background-image: url("bg.jpg");
  background-repeat: no-repeat;
  background-attachment: local;
}
```

.2 CSS Gradient Backgrounds

A color gradient consists of two or more colors that blend, transitioning from one color to another over a given distance.

We will use this HTML file in all the examples in this section:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

'.2.1 Linear Gradients

Below is the syntax of the linear gradients:

```
background: linear-gradient(direction, color-stop1, color-stop2, ...);
```

Linear gradient “top to bottom” is the default, so we can declare it like this:

Example 1:

style.css

```
p {
  height: 100px;
  background: linear-gradient(red, yellow);
}
```

Result:

Linear gradient “left to right”:

Example 2:

style.css

```
p {
  height: 100px;
  background: linear-gradient(to right, red, yellow);
}
```

Result:

Diagonal linear gradient:

Example 3:

style.css

```
p {
  height: 100px;
  background: linear-gradient(to bottom right, red, yellow);
}
```

Result:

We can specify an angle of linear gradients using this syntax:

```
background: linear-gradient(angle, color-stop1, color-stop2);
```

Example 4:

style.css

```
p {
  height: 100px;
  background: linear-gradient(-90deg, red, yellow);
}
```

Result:

'.2.2 Repeating Linear Gradients

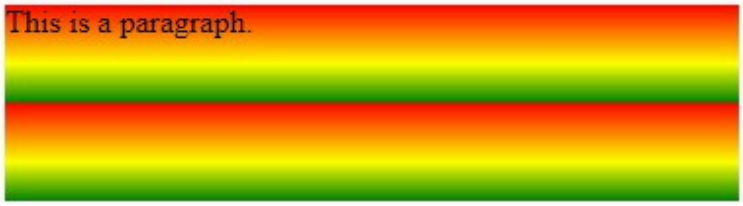
The " repeating-linear-gradient " property helps to repeat a linear gradient.

Example:

style.css

```
p {
  height: 100px;
  background: repeating-linear-gradient(red, yellow 30%, green 50%);
}
```

Result:



The above CSS is a repeating gradient going from top to bottom, starting red, turning yellow after 30%, and finishing green at 50%. Therefore, we will see it is repeated two times.

'.2.3 Radial Gradients

Below is the syntax of the linear gradients:

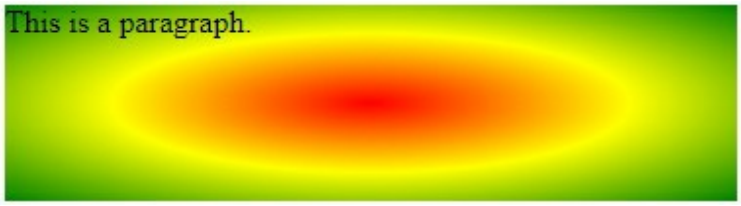
```
background: radial-gradient(shape size at position, start-color, ..., last-color);
```

Example 1: Radial gradient with evenly spaced color stops (this is the default).

style.css

```
p {
  height: 100px;
  background: radial-gradient(red, yellow, green);
}
```

Result:

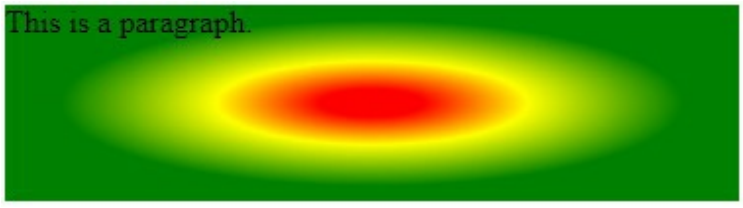


Example 2: Radial gradient with differently spaced color stops.

style.css

```
p {
  height: 100px;
  background: radial-gradient(red 10%, yellow 30%, green 60%);
}
```

Result:



The above example creates a gradient at the center of its container, starting red, changing to yellow from 10% to 30%, and finishing green from 60%.

'.2.4 Repeating Radial Gradients

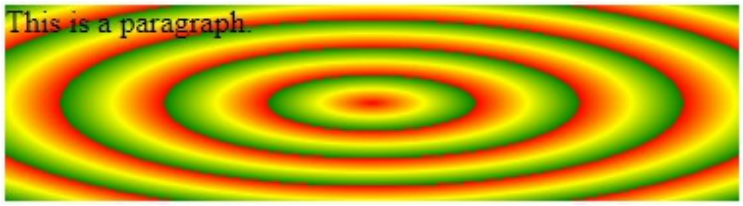
The “ repeating-radial-gradient ” property helps to repeat a radial gradient.

Example:

style.css

```
p {
  height: 100px;
  background: repeating-radial-gradient(red, yellow 10%, green 20%);
}
```

Result:



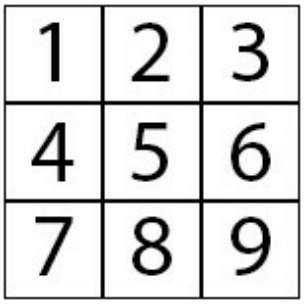
The above example creates a repeating gradient at the center of its container, starting red, changing to yellow from 10%, and finishing green at 20%. Therefore, we will see it repeated five times.

.3 CSS Sprites

To reduce HTTP requests to the server, we can merge many images into a single image called image sprite.

CSS sprite only works with block or inline-block elements since we cannot set the height of the inline elements.

In the below example, we will use this image (150x150 pixels). Therefore, each number has a size of 50x50 pixels.



Example 1:

index.html

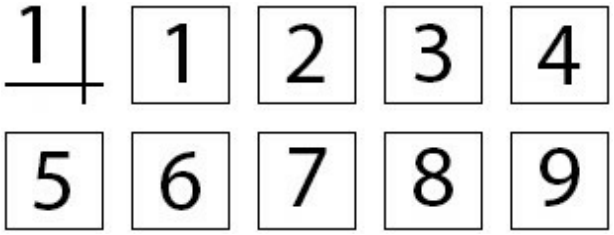
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="sprite testing"></div>
    <div class="sprite number-1"></div>
    <div class="sprite number-2"></div>
    <div class="sprite number-3"></div>
    <div class="sprite number-4"></div>
    <div class="sprite number-5"></div>
    <div class="sprite number-6"></div>
    <div class="sprite number-7"></div>
    <div class="sprite number-8"></div>
    <div class="sprite number-9"></div>
  </body>
```

```
</html>
```

style.css

```
.sprite {
  background-image: url("sprites.jpg");
  height: 50px;
  width: 50px;
  margin: 5px;
  display: inline-block;
}

.testing {
  background-position: -10px -10px;
}
.number-1 {
  background-position: 0 0;
}
.number-2 {
  background-position: -50px 0;
}
.number-3 {
  background-position: -100px 0;
}
.number-4 {
  background-position: 0 -50px;
}
.number-5 {
  background-position: -50px -50px;
}
.number-6 {
  background-position: -100px -50px;
}
.number-7 {
  background-position: 0 -100px;
}
.number-8 {
  background-position: -50px -100px;
}
.number-9 {
  background-position: -100px -100px;
}
```

Result:



It can be seen that CSS sprites use the background-position property to shift a portion of a big image into a smaller viewport like the element.

We often use the image sprite in the navigation bar like this.

Example 2:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <ul class="navlist">
   <li class="sprite home"><a href="#home"></a></li>
   <li class="sprite prev"><a href="#prev"></a></li>
   <li class="sprite next"><a href="#next"></a></li>
  </ul>
 </body>
</html>
```

style.css

```
.navlist li {
 margin: 0;
 padding: 0;
 list-style: none;
 float: left;
}
.navlist li,
.navlist a {
 display: block;
 height: 50px;
 width: 50px;
}
```

```
.sprite {
  background-image: url("sprites.jpg");
}

.home {
  background-position: 0 0;
}
.home:hover {
  background-position: 0 -50px;
}

.prev {
  background-position: -50px 0;
}
.prev:hover {
  background-position: -50px -50px;
}

.next {
  background-position: -100px 0;
}
.next:hover {
  background-position: -100px -50px;
}
```

CHAPTER 18
CSS Display

.1 display vs. visibility

3.1.1 display

The “ display ” property specifies if/how an element is displayed.

“ display: none; ” is commonly used with JavaScript to hide or show elements without deleting and recreating them. For example, to show a hidden element, we could set it to “ display: block ,”“ display: inline ” or “ display: inline-block ” according to the HTML element type.

The CSS “ height ” property will not affect the inline elements. Therefore, to convert a block element into an inline but keep its height, we use “ display: inline-block; .”

The inline-block elements have both characteristics of inline and block elements. They only occupy the space necessary for their contents so that they will not cause any line breaks. However, they could have more space than their contents by using these CSS properties: width, height, padding, and margin .

Example:

index.html

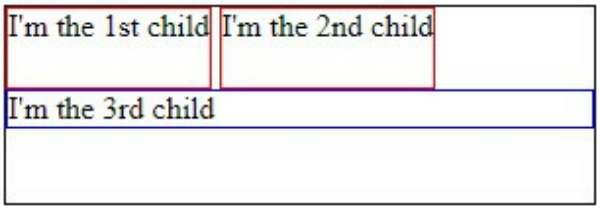
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
```

```
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div class="inline-blocks">I'm the 1st child</div>
      <div class="inline-blocks">I'm the 2nd child</div>
      <div class="normal-blocks">I'm the 3rd child</div>
    </div>
  </body>
</html>
```

style.css

```
.container {
  border: 1px solid black;
  width: 300px;
  height: 100px;
}

.inline-blocks {
  display: inline-block;
  border: 1px solid red;
  height: 40px;
}

.normal-blocks {
  border: 1px solid blue;
}
```

Result:



This example is another way to make elements float without the “ float ” property.

3.1.2 visibility

To hide the element but still take up the same space of it as before, we use the “ visibility ” property.

Example:

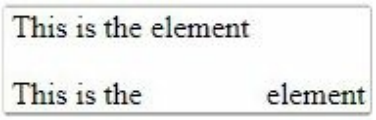
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>This is the <span class="hidden">hidden</span> element</p>
    <p>This is the <span class="invisible">invisible</span> element</p>
  </body>
</html>
```

style.css

```
.hidden {
  display: none;
}

.invisible {
  visibility: hidden;
}
```

Result:



.2 CSS object-fit

The CSS “ object-fit ” property is used to specify how the object (image or video) should be resized to fit its container (the <img> or <video> elements). The property can have the following values:

- fill (default value) - The object is resized to fill the element's content box. If necessary, the object will be distorted to fit.
- contain - The object is scaled to maintain its aspect ratio while fitting within the element's content box.
- cover - The object is scaled to maintain its aspect ratio while filling the element's entire content box. The object will be clipped to fit.
- none - The object is not resized.
- scale-down - The object is sized as “ none ” or “ contain ” if the element's content box is smaller than the object size.

In the below examples, we will use an image of 150x150 pixels.

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <img src="bg.jpg" />
  </body>
</html>
```

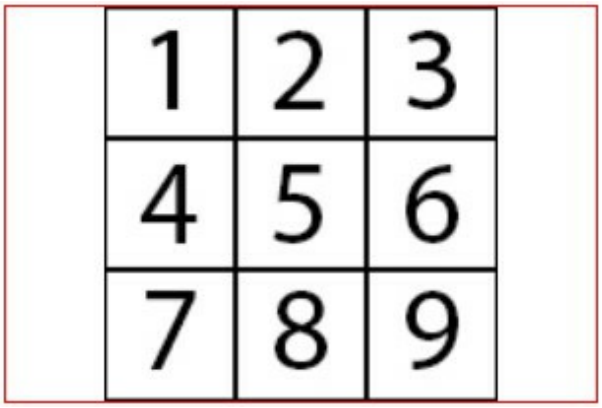
Example 1:

style.css

```
img {
  width: 300px;
  height: 200px;
  border: 1px solid red;
  object-fit: contain;
```

```
}
```

Result:

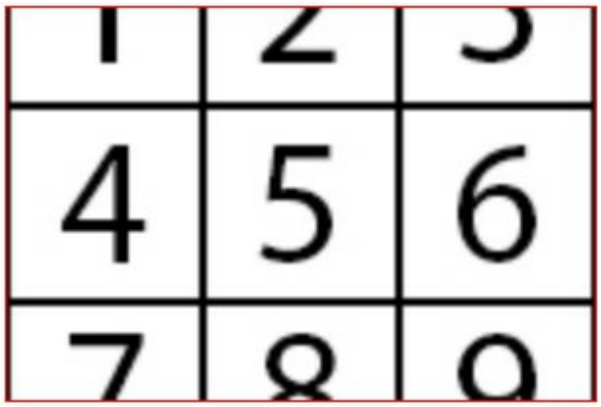


Example 2:

style.css

```
img {
  width: 300px;
  height: 200px;
  border: 1px solid red;
  object-fit: cover;
}
```

Result:



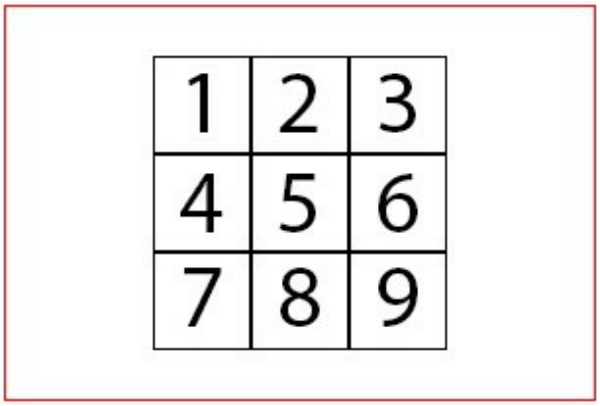
Example 3:

style.css

```
img {
  width: 300px;
  height: 200px;
```

```
  border: 1px solid red;
  object-fit: none;
}
```

Result:

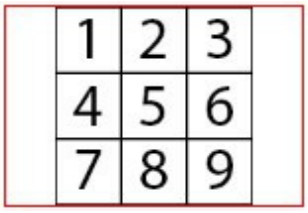


Example 4:

style.css

```
img {
  width: 150px;
  height: 100px;
  border: 1px solid red;
  object-fit: scale-down;
}
```

Result:



.3 Cursor

The “ cursor ” property specifies how the mouse’s pointer looks when the user hovers it on an element. These are popular values of this property:

- pointer - The cursor is a pointer
- not-allowed - The cursor indicates that the requested action will not be executed
- grab - The cursor indicates that something can be grabbed
- grabbing - The cursor indicates that something is grabbing
- help - The cursor indicates that help is available
- move - The cursor indicates something is to be moved
- zoom-in - The cursor indicates that something can be zoomed in
- zoom-out - The cursor indicates that something can be zoomed out

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p class="my-element">This is a paragraph</p>
  </body>
</html>
```

style.css

```
.my-element {
  cursor: pointer;
}
```

.4 CSS Overflow

The overflow property specifies how the overflowed content is rendered inside its containing element.

This file will be used in the following examples:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div class="child">I am the content.</div>
    </div>
  </body>
</html>
```

3.4.1 overflow: visible

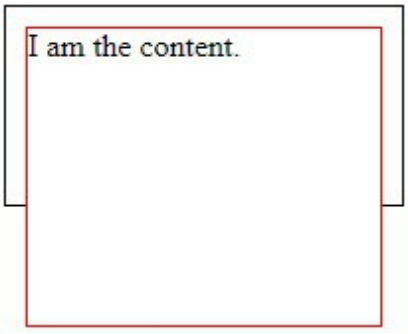
By default, the overflow is visible. It means that the content will be rendered outside its parent.

Example:

style.css

```
.container {
  border: 1px solid black;
  width: 200px;
  height: 100px;
}

.child {
  margin: 10px;
  border: 1px solid red;
  background-color: white;
  height: 150px;
}
```

Result:



3.4.2 overflow: hidden

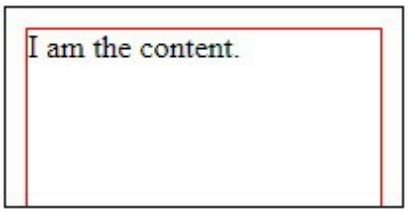
With the “ hidden ” value, the overflowed content will be hidden.

Example:

style.css

```
.container {
  overflow: hidden;
  border: 1px solid black;
  width: 200px;
  height: 100px;
}

.child {
  margin: 10px;
  border: 1px solid red;
  background-color: white;
  height: 150px;
}
```

Result:



3.4.3 overflow: scroll

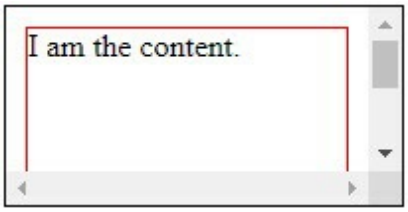
Setting the value to “ scroll ,” horizontally and vertically scrollbars are added to view the overflowed content. The two scrollbars are always shown despite needing.

Example:

style.css

```
.container {
  overflow: scroll;
  border: 1px solid black;
  width: 200px;
  height: 100px;
}

.child {
  margin: 10px;
  border: 1px solid red;
  background-color: white;
  height: 150px;
}
```

Result:



3.4.4 overflow: auto

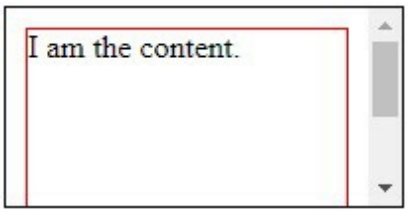
The “ auto ” value will add scrollbars when necessary.

Example:

style.css

```
.container {
  overflow: auto;
  border: 1px solid black;
  width: 200px;
  height: 100px;
}

.child {
  margin: 10px;
  border: 1px solid red;
  background-color: white;
  height: 150px;
}
```

Result:



.5 CSS Opacity

The “ opacity ” property sets the transparency of an element and all its children, including backgrounds and texts. Its valid value is a float from 0.0 to 1.0, where 0 is completely transparent and 1 is not transparent.

The example below also compares the “ opacity ” property and the transparent“ background-color .” Again, we use the transparent background color to keep the text unchanged.

Example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="container">
   <div class="child">I'm a child element</div>
  </div>
  <div class="container opacity">
   <div class="child">I'm a child element</div>
  </div>
  <div class="container transparent">
   <div class="child">I'm a child element</div>
  </div>
 </body>
</html>
```

style.css

```
.container {
 background-color: rgb(255, 0, 0);
 padding: 20px;
 margin: 10px;
}

.container.opacity {
 opacity: 0.5;
```

```
}

.container.transparent {
  background-color: rgba(255, 0, 0, 0.5);
}

.child {
  border: 1px solid black;
}
```

Result:

.6 CSS Shadows

We can add shadows to text and HTML elements using CSS.

.6.1 text-shadow

The " text-shadow " property adds a shadow to text and accepts a comma-separated list of shadows. The syntax is:

```
text-shadow: h-offset v-offset blur-radius color
```

where:

- h-offset - The horizontal position of the shadow (allows negative values to put the shadow on the left side of the text).
- v-offset - The vertical position of the shadow (allows negative values to put the shadow above the text).
- blur-radius (optional) - The higher the number, the more blurred the shadow will be.
- color (optional) - The color of the shadow (The default value is the text color).

Example 1:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>This is a heading</h1>
  </body>
</html>
```

style.css

```
h1 {
  text-shadow: 3px 3px red;
```

```
}
```

Result:

Example 2: To make it real, we should put a blur-radius into the CSS at the 3rd position.

```
h1 {
  text-shadow: 3px 3px 5px red;
}
```

Result:

Example 3: Multiple shadows

To add more than one shadow to the text, you can add a comma-separated list of shadows.

```
h1 {
  color: white;
  text-shadow: 0 0 2px #f00, 0 0 20px #00f;
}
```

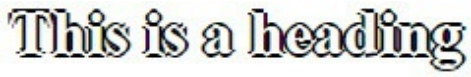
Result:

Example 4:

The below CSS creates a 2px border around the text. The solution is straightforward, so we create four shadows (without blur) and then shift them left (-2px 0 black), bottom (0 2px black), right (2px 0 black), and top (0 -2px black), each side 2px.

```
h1 {
  color: white;
  text-shadow: -2px 0 black, 0 2px black, 2px 0 black, 0 -2px black;
}
```

Result:



3.6.2 box-shadow

The " box-shadow " property adds one or more shadows to an element. The syntax is:

```
box-shadow: h-offset v-offset blur-radius spread color
```

where:

- h-offset - The horizontal position of the shadow (allows negative values to put the shadow on the left side of the text).
- v-offset - The vertical position of the shadow (allows negative values to put the shadow above the text).
- blur-radius (optional) - The higher the number, the more blurred the shadow will be.
- spread (optional) - A positive value increases the size of the shadow; a negative value decreases the size of the shadow.
- color (optional) - The color of the shadow (The default value is the text color).

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>This is a div</div>
  </body>
</html>
```

style.css

Example 1:

```
div {
  border: 1px solid black;
  box-shadow: 5px 5px grey;
}
```

Result:

Example 2: To make it real, we should put a blur number into the CSS at the 3rd position.

```
div {
  border: 1px solid black;
  box-shadow: 5px 5px 5px grey;
}
```

Result:

Example 3: Let's make the shadow bigger by adding a spread number into the CSS at the 4th position.

```
div {
  border: 1px solid black;
  box-shadow: 5px 5px 5px 2px grey;
}
```

Result:

Example 4: To add more than one shadow to the text, you can add a comma-separated list of shadows.

```
div {
  border: 1px solid black;
  box-shadow: 8px 0 10px grey, 0 8px 10px grey;
}
```

Result:

This is a div

.7 CSS Multiple Columns

The CSS multi-column layout allows easy definition of multiple columns of text - like the newspapers.

3.7.1 Fixed Columns

These CSS properties help us define a multi-column layout:

- The column-count property specifies the number of columns an element should be divided into.
- The column-gap property specifies the gap between the columns.
- The column-rule-style property specifies the style of the rule between columns.
- The column-rule-width property specifies the width of the rule between columns.
- The column-rule-color property specifies the color of the rule between columns.

Example 1:

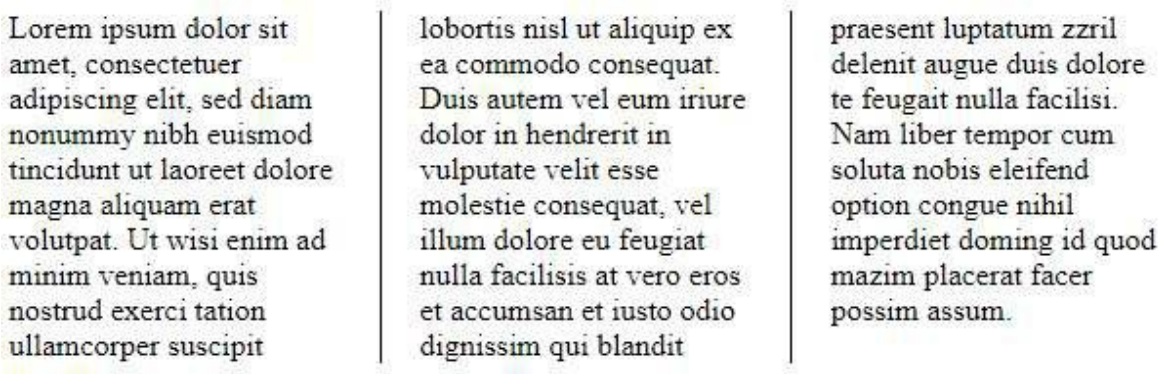
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="newspaper">Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.</div>
  </body>
</html>
```

style.css

```
.newspaper {
  column-count: 3;
  column-gap: 40px;
  column-rule-style: solid;
  column-rule-width: 1px;
  column-rule-color: black;
}
```

Result:



The column-rule property is a shorthand property for setting all the column-rule-* properties above.

```
.newspaper {
  column-count: 3;
  column-gap: 40px;
  column-rule: 1px solid black;
}
```

3.7.2 Responsive Columns

The “ column-width ” property specifies a suggested, optimal width for the columns.

```
.newspaper {
  column-count: 3;
  column-gap: 40px;
  column-rule: 1px solid black;
  column-width: 200px;
}
```

Try to resize the browser to see how the number of columns will change

based on the viewport’s width.

3.7.3 Column Span

The “ column-span ” property specifies whether an HTML element should span across all the columns by using “ column-span: all .”

Example 2:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="newspaper">
   <h1>Lorem Ipsum Dolor Sit Amet</h1>
   <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.</p>
  </div>
 </body>
</html>
```

style.css

```
.newspaper {
 column-count: 3;
 column-gap: 40px;
 column-rule: 1px solid black;
}

h1 {
 column-span: all;
}
```

Result:

Lorem Ipsum Dolor Sit Amet

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.

.8 CSS Tables

Below is a simple table.

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <table>
   <thead>
    <tr>
     <th>Firstname</th>
     <th>Lastname</th>
    </tr>
   </thead>
   <tbody>
    <tr>
     <td>John</td>
     <td>Corner</td>
    </tr>
    <tr>
     <td>Neo</td>
     <td>D. Truman</td>
    </tr>
   </tbody>
  </table>
 </body>
</html>
```

Example 1:

style.css

```
table {
 border: 1px solid green;
}

th {
 border: 1px solid red;
```

```
}

td {
  border: 1px solid blue;
}
```

Result:

Firstname	Lastname
John	Conner
Neo	D. Truman

We see double borders because of the space between cells inside the table.

The “ border-collapse ” property sets whether the table borders should be collapsed into a single border.

Example 2:

style.css

```
table {
  border-collapse: collapse;
  border: 1px solid green;
}

th {
  border: 1px solid red;
}

td {
  border: 1px solid blue;
}
```

Result:

Firstname	Lastname
John	Conner
Neo	D. Truman

3.8.1 Table Sizing

We can set the table's width and height or its cells' width and height using " width " and " height " properties. For example:

style.css

```
table {
  border-collapse: collapse;
  border: 1px solid green;
  width: 100%;
}

th {
  border: 1px solid red;
  height: 30px;
}

td {
  border: 1px solid blue;
  height: 50px;
}
```

Result:

Firstname	Lastname
John	Conner
Neo	D. Truman

3.8.2 Text Alignment in Table Cells

We can align text inside cells using " text-align " (with value: left, right, center, justify) and " vertical-align " (with value: baseline, length, sub, super, top, text-top, middle, bottom, text-bottom) properties. For example:

style.css

```
table {
  border-collapse: collapse;
  border: 1px solid green;
  width: 100%;
}

th {
  border: 1px solid red;
  height: 30px;
}

td {
  border: 1px solid blue;
  height: 50px;
  text-align: center;
  vertical-align: top;
}
```

Result:

Firstname	Lastname
John	Conner
Neo	D. Truman

.9 CSS Lists

In HTML, there are two main types of lists:

- unordered lists ( ) - the list items are marked with bullets.
- ordered lists ( ) - the list items are marked with numbers or letters.

.9.1 List Item Markers

The “ list-style-type ” property specifies the type of list item marker. The value of this property can be circle, square, decimal, decimal-leading-zero, lower-alpha, lower-roman, upper-alpha, upper-roman, or none .

Example 1:

index.html

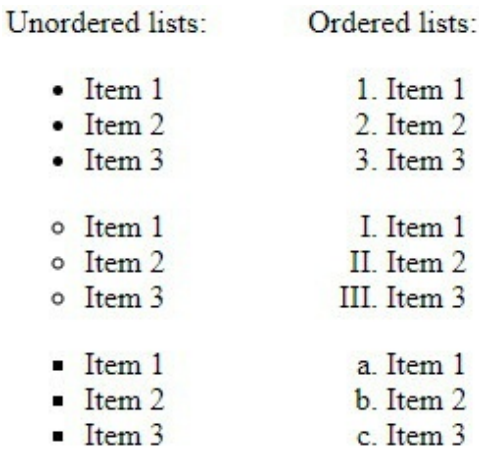
```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="col">
   <p>Unordered lists:</p>
   <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
   </ul>
   <ul class="circle">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
   </ul>
   <ul class="square">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
   </ul>
```

```
    </div>
    <div class="col">
      <p>Ordered lists:</p>
      <ol>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
      </ol>
      <ol class="upper-roman">
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
      </ol>
      <ol class="lower-alpha">
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
      </ol>
    </div>
  </body>
</html>
```

style.css

```
.col {
  display: inline-block;
  width: 150px;
}

ul.circle {
  list-style-type: circle;
}
ul.square {
  list-style-type: square;
}

ol.upper-roman {
  list-style-type: upper-roman;
}
ol.lower-alpha {
  list-style-type: lower-alpha;
}
```

Result:



The " list-style-image " property specifies an image as the list item marker.

Example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <ul>
   <li>Item 1</li>
   <li>Item 2</li>
   <li>Item 3</li>
  </ul>
 </body>
</html>
```

style.css

```
ul {
 list-style-image: url("marker.png");
}
```

3.9.2 Position the List Item Markers

The " list-style-position " property specifies whether the list-item markers should appear inside or outside the content flow.

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ul>
  </body>
</html>
```

style.css

```
ul {
  list-style-position: inside;
}
```

This shorthand list-style property simultaneously sets all three list-style properties(list-style-type, list-style-position, list-style-image).

Example:

```
ul {
  list-style: upper-latin inside;
}
```

.10 CSS Transforms

.10.1 Moves an Element

We can move an HTML element to another position by using one of these CSS declarations:

- transform: **translateX(x)** - Moves the element along the X-axis.
- transform: **translateY(y)** - Moves the element along the Y-axis.
- transform: **translateZ(z)** - Moves the element along the Z-axis.
- transform: **translate(x,y)** - Moves the element along the X-axis and Y-axis.
- transform: **translate3d(x,y,z)** - Moves the element along the three axes.

The following example moves the element 30 pixels to the right and 60 pixels down from its original position.

index.html

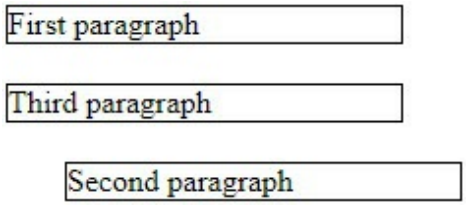
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>First paragraph</div>
    <div class="test">Second paragraph</div>
    <div>Third paragraph</div>
  </body>
</html>
```

style.css

```
div {
  width: 200px;
  border: 1px solid black;
}
```

```
.test {
  transform: translate(30px, 60px);
}
```

Result:



3.10.2 Scales an Element

We can scale an HTML element by using one of these CSS declarations:

- transform: **scaleX(x)** - Scales the element along the X-axis.
- transform: **scaleY(y)** - Scales the element along the Y-axis.
- transform: **scaleZ(z)** - Scales the element along the Z-axis.
- transform: **scale(x,y)** - - Scales the element along the X-axis and Y-axis.
- transform: **scale3d(x,y,z)** - Scales the element along the three axes.

The following example increases the element to two times its original width and three times its original height.

Example:

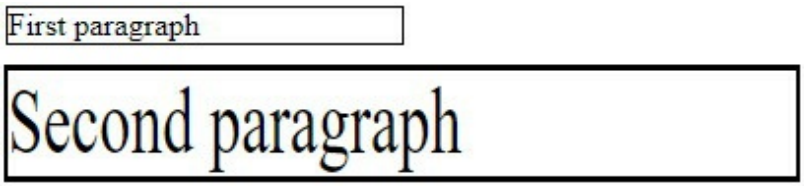
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>First paragraph</div>
    <div class="test">Second paragraph</div>
  </body>
</html>
```

style.css

```
div {
  width: 200px;
  border: 1px solid black;
}

.test {
  position: relative;
  left: 100px;
  top: 30px;
  transform: scale(2, 3);
}
```

Result:



3.10.3 Rotates an Element

We can rotate an HTML element by using one of these CSS declarations:

- transform: rotateX(angle) - Defines a rotation along the X-axis.
- transform: rotateY(angle) - Defines a rotation along the Y-axis.
- transform: rotateZ(angle) - Defines a rotation along the Z-axis.
- transform: rotate(angle) - Defines a 2D rotation at a given angle.
- transform: rotate3d(x,y,z,angle) - Defines a 3D rotation.

Example 1:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>First paragraph</div>
    <div class="div2">Second paragraph</div>
    <div class="div3">Third paragraph</div>
  </body>
</html>
```

style.css

```
.div2 {
  transform: rotate(180deg);
}

.div3 {
  transform: rotate(-20deg);
}
```

Result:

The rotateX(), rotateY() , or rotateZ() method rotates an element around its X-axis, Y-axis, or Z-axis at a given degree.

Example 2:

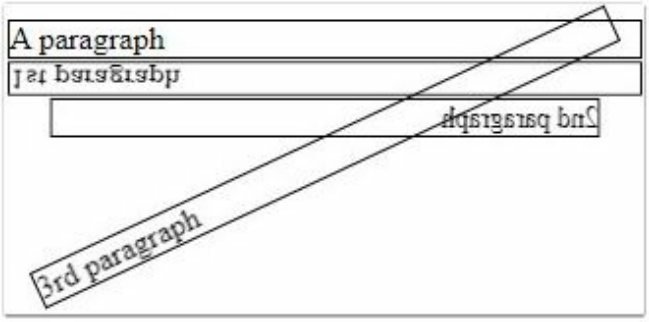
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>A paragraph</div>
    <div class="test1">1st paragraph</div>
    <div class="test2">2nd paragraph</div>
    <div class="test3">3rd paragraph</div>
  </body>
</html>
```

style.css

```
div {
  border: 1px solid black;
}

.test1 {
  transform: rotateX(150deg);
}
.test2 {
  transform: rotateY(150deg);
}
.test3 {
  transform: rotateZ(-100deg);
}
```

Result:



3.10.4 Skews an Element

We can skew an HTML element by using one of these CSS declarations:

- transform: skewX(angle) - Skews an element along the X-axis.
- transform: skewY(angle) - Skews an element along the Y-axis.
- transform: skew(x-angle,y-angle) - Defines a 2D skew transformation along the X-axis and Y-axis.

Example:

index.html

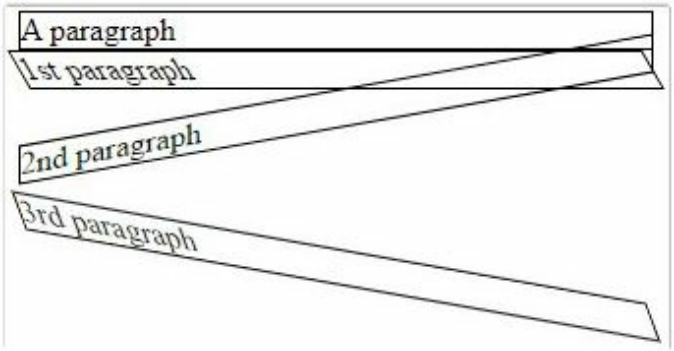
```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div>A paragraph</div>
  <div class="test1">1st paragraph</div>
  <div class="test2">2nd paragraph</div>
  <div class="test3">3rd paragraph</div>
 </body>
</html>
```

style.css

```
div {
 border: 1px solid black;
}

.test1 {
 transform: skewX(30deg);
```

```
}
.test2 {
  transform: skewY(-10deg);
}
.test3 {
  position: relative;
  top: 60px;
  transform: skew(20deg, 10deg);
}
```

Result:



CHAPTER 19
CSS Layout

The layout is the way HTML elements are placed on a web page. We often use CSS to build a design by arranging the elements into our preferred visual structure.

.1 CSS Position

9.1.1 static

The default value of the CSS “ position ” property is “ static .” The element is placed where it should be inside the web page according to its order in the HTML code. We said that the element is not positioned.

9.1.2 relative

An element with “ position: relative ” is relative to its normal position. We use the top, right, bottom , and left properties to set the element’s position.

index.html

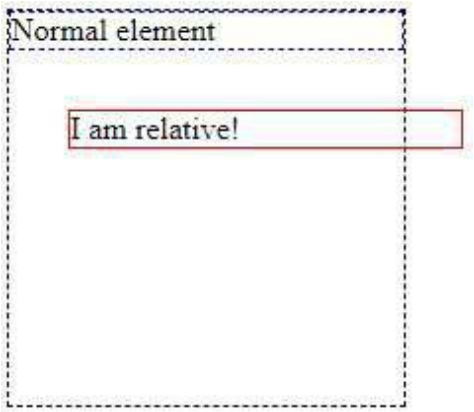
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="parent">
      <div class="normal">Normal element</div>
      <div class="relative">I am relative!</div>
    </div>
  </body>
```

```
</html>
```

style.css

```
.parent {
  width: 200px;
  height: 200px;
  border: 1px dashed black;
}

.relative {
  position: relative;
  left: 30px;
  top: 30px;
  border: 1px solid red;
}

.normal {
  border: 1px dashed blue;
}
```

Result:



9.1.3 absolute

An element with “ position: absolute ” is positioned relative to the nearest positioned ancestor.

We can set the“ position ” property of the parent element to “ relative ” to make it a positioned element without changing its position.

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="positioned">
      <div class="non-positioned">
        <div class="absolute">I am absolute!</div>
      </div>
    </div>
  </body>
</html>
```

style.css

```
.positioned {
  position: relative;
  border: 1px solid black;
  width: 200px;
  height: 200px;
}

.non-positioned {
  border: 1px dashed black;
  width: 150px;
  height: 150px;
}
```

```
.absolute {
  position: absolute;
  right: 30px;
  top: 30px;
  border: 1px solid red;
  background-color: white;
}
```

Result: The element is positioned relative to the element since this is the nearest positioned ancestor.

9.1.4 fixed

An element with " position: fixed " is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled.

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>Normal element</div>
    <div class="fixed">I am fixed!</div>
    <div class="big">A big element.</div>
  </body>
```

```
</html>
```

style.css

```
.big {
  height: 1000px;
  border: 1px solid black;
}

.fixed {
  position: fixed;
  right: 30px;
  top: 30px;
  width: 100px;
  border: 1px solid red;
}
```

9.1.5 sticky

An element with “ position: sticky ” is left in the normal flow until the conditions that trigger its stickiness come to pass.

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="parent">
      <p>Try to scroll inside this frame to understand how sticky positioning works.</p>
      <div class="sticky">I am sticky!</div>
      <p>In this example, the sticky element sticks to the top of the page when it reaches the position "top: 0".</p>
      <p>Scroll back up to remove the stickiness.</p>
      <p>Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo.</p>
    </div>
  </body>
</html>
```

style.css

```css
.parent {
  width: 200px;
  height: 200px;
  overflow: auto;
}

.sticky {
  position: sticky;
  top: 0;
  border: 1px solid red;
  background-color: white;
}
```

.2 CSS Float

9.2.1 float

Make elements float inside their containing element. There are several use cases:

- “ float: none ” (default value) - Elements are placed in the normal flow.
- “ float: left ” - Elements float on the left side of their parent.
- “ float: right ” - Elements float on the right side of their parent but in reverted order.

Example:

index.html

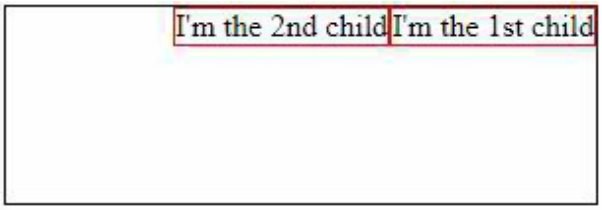
```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="container">
   <div class="child">I'm the 1st child</div>
   <div class="child">I'm the 2nd child</div>
  </div>
 </body>
</html>
```

style.css

```
.container {
 border: 1px solid black;
 width: 300px;
 height: 100px;
}

.child {
 float: right;
 border: 1px solid red;
```

```
}
```

Result:



9.2.2 clear

The example below shows us that the 3rd element is not placed in the expected place.

Example:

index.html

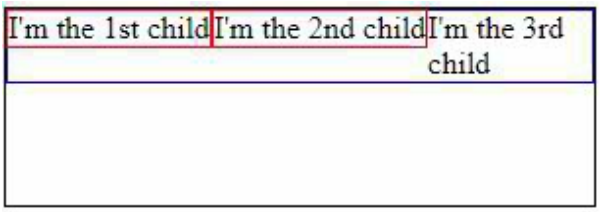
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div class="float">I'm the 1st child</div>
      <div class="float">I'm the 2nd child</div>
      <div class="no-float">I'm the 3rd child</div>
    </div>
  </body>
</html>
```

style.css

```
.container {
  border: 1px solid black;
  width: 300px;
  height: 100px;
}

.float {
  float: left;
  border: 1px solid red;
```

```
}

.no-float {
  border: 1px solid blue;
}
```

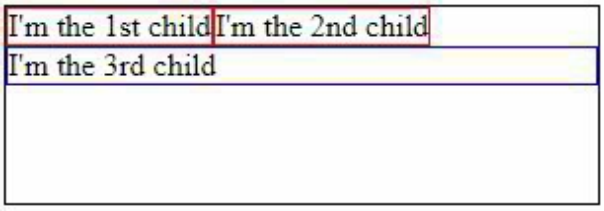
Result:



To force an element to start below any floated elements, we use the “ clear ” property. In this case, we use “ clear: left ” because the above elements are floated left.

```
.no-float {
  clear: left;
  border: 1px solid blue;
}
```

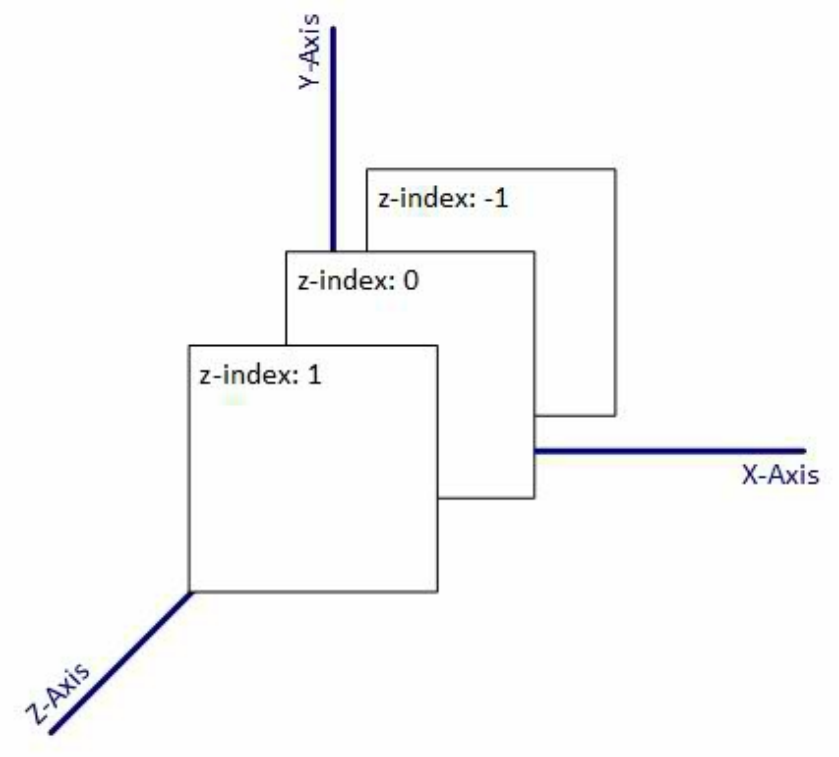
Result:



Two other options of the “ clear ” property are “ clear: right; ” or “ clear: both; .”

.3 CSS Z-index

The “ z-index ” property specifies the order of positioned elements on the Z-axis.



Example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="container">
   <div class="absolute child z-index-minus-two">z-index: -2</div>
   <div class="absolute child z-index-minus-one">z-index: -1</div>
   <div class="absolute child z-index-zero">z-index: 0</div>
   <div class="absolute child z-index-one">z-index: 1</div>
   <div class="child z-index-two">z-index: 2; but I'm not positioned</div>
  </div>
```

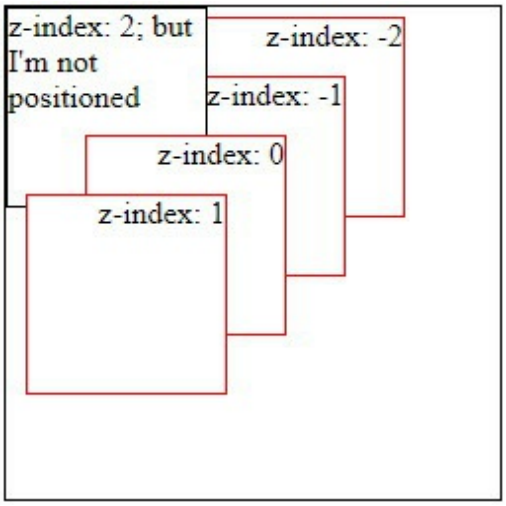
```
  </body>
</html>
```

style.css

```
div {
  border: 1px solid black;
}

.container {
  position: relative;
  width: 250px;
  height: 250px;
}

.child {
  width: 100px;
  height: 100px;
  background-color: white;
}

.absolute {
  position: absolute;
  border: 1px solid red;
  text-align: right;
}

.z-index-minus-two {
  z-index: -2;
  left: 100px;
  top: 5px;
}

.z-index-minus-one {
  z-index: -1;
  left: 70px;
  top: 35px;
}

.z-index-zero {
  z-index: 0;
  left: 40px;
  top: 65px;
}
```

```
.z-index-one {
  z-index: 1;
  left: 10px;
  top: 95px;
}

.z-index-two {
  z-index: 2;
}
```

Result:



.4 CSS Flexbox

A flexible box or flexbox ensures that elements behave predictably when the page layout must accommodate different screen sizes and display devices. A flex container is declared by setting the “ display ” property of an element to either “ flex ” (rendered as a block) or “ inline-flex ” (rendered as inline).

Inside a flex container, there are one or more flex items. Flex items are positioned inside a flex container along a flex line. By default, there is only one flex line per flex container.

The following example shows three flex items. They are positioned by default: along the horizontal flex line from left to right.

Example:

index.html

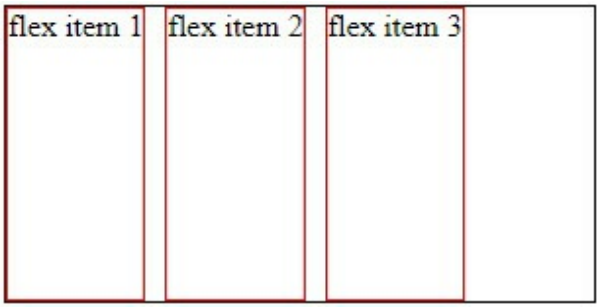
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item item1">flex item 1</div>
      <div class="flex-item item2">flex item 2</div>
      <div class="flex-item item3">flex item 3</div>
    </div>
  </body>
</html>
```

style.css

```
.flex-container {
  display: flex;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}
```

```
.flex-item {
  border: 1px solid red;
}
```

Result:



9.4.1 Flex Container

9.4.1.1 gap

The “ gap ” property is used to create space between flex items. As in the previous example, the gap between flex items is 10 pixels.

9.4.1.2 flex-direction

The “ flex-direction ” property specifies the direction of the flexible items inside the flex container. The default value of flex-direction is row (left-to-right).

The other values are as follows:

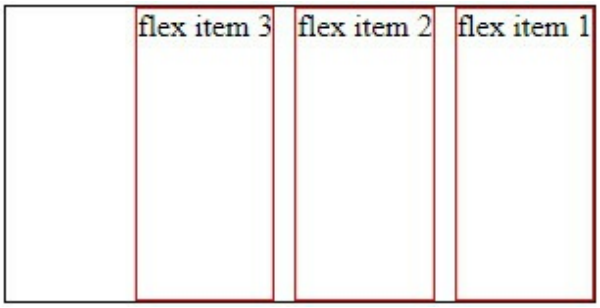
- row - The flex items are displayed horizontally as a row.
- row-reverse - Same as row, but the flex items will be laid out right to the left.
- column - The flex items are displayed vertically as a column.
- column-reverse - Same as the column, but in reverse order.

Example 1:

style.css

```
.flex-container {
  display: flex;
  flex-direction: row-reverse;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:



Example 2:

style.css

```
.flex-container {
  display: flex;
  flex-direction: column;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

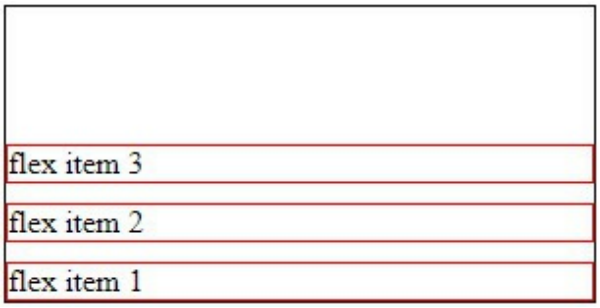
Example 3:

style.css

```
.flex-container {
  display: flex;
  flex-direction: column-reverse;
  width: 300px;
  height: 150px;
  border: 1px solid black;
```

```
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:



9.4.1.3 *justify-content*

The “ justify-content ” property horizontally aligns the flexible container's items when the items do not use all available space.

The possible values are as follows:

- flex-start (default value) - Items are positioned at the beginning of the container.
- flex-end - Items are positioned at the end of the container.
- center - Items are positioned at the center of the container.
- space-between - Items are positioned with space between the lines.
- space-around - Items are positioned with space before, between, and after the lines.
- space-evenly - Items will have equal space around them.

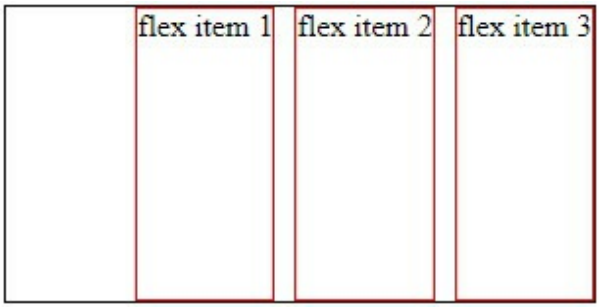
Example 1:

style.css

```
.flex-container {
  display: flex;
  justify-content: flex-end;
  width: 300px;
  height: 150px;
```

```
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

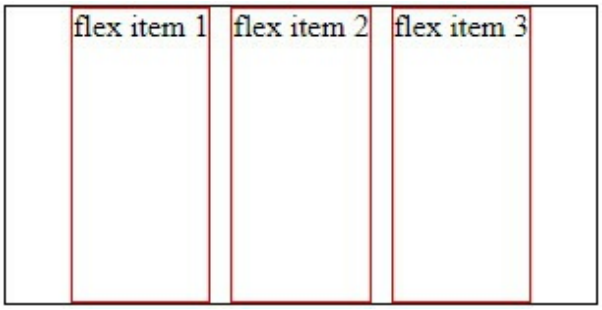


Example 2:

style.css

```
.flex-container {
  display: flex;
  justify-content: center;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

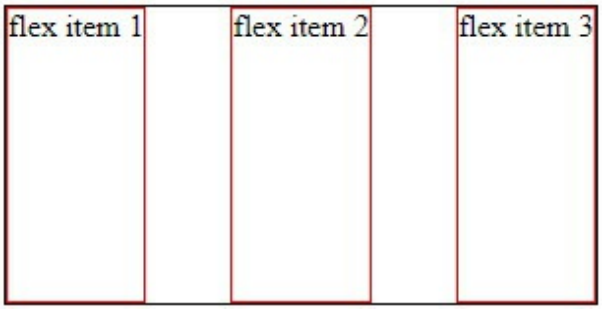


Example 3:

style.css

```
.flex-container {
  display: flex;
  justify-content: space-between;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

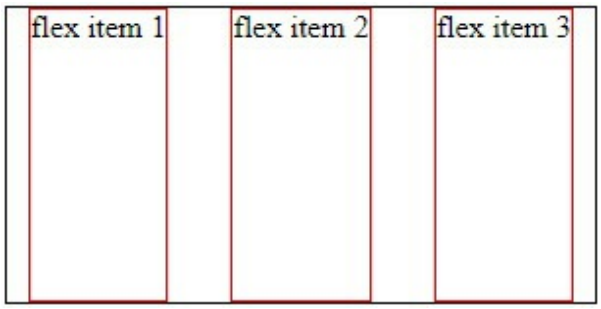


Example 4:

style.css

```
.flex-container {
  display: flex;
  justify-content: space-around;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

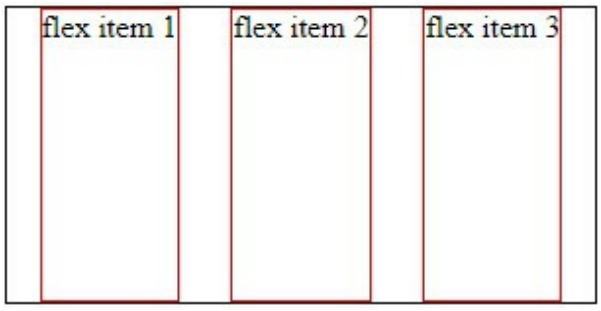


Example 5:

style.css

```
.flex-container {
  display: flex;
  justify-content: space-evenly;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:



9.4.1.4 *align-items*

The “ align-items ” property vertically aligns the flexible container's items when the items do not use all available space.

The possible values are as follows:

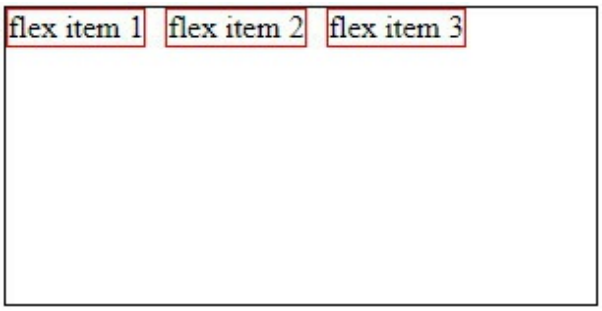
- stretch (default value) - Items are stretched to fit the container
- flex-start - Items are positioned at the top of the container
- flex-end - Items are positioned at the bottom of the container
- center - Items are positioned at the center of the container (vertically)
- baseline - Items are aligned along the text baseline of the container

Example 1:

style.css

```
.flex-container {
  display: flex;
  align-items: flex-start;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

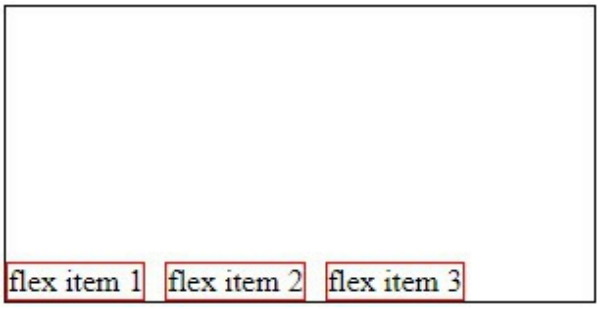


Example 2:

style.css

```
.flex-container {
  display: flex;
  align-items: flex-end;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

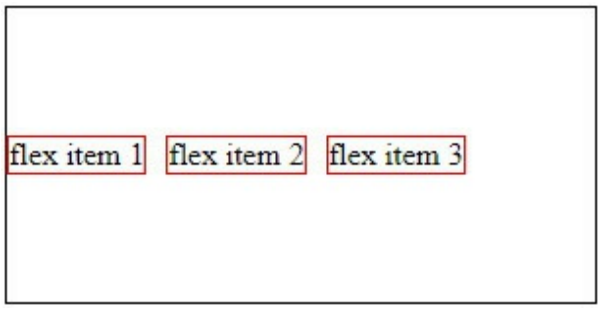


Example 3:

style.css

```
.flex-container {
  display: flex;
  align-items: center;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}
```

Result:

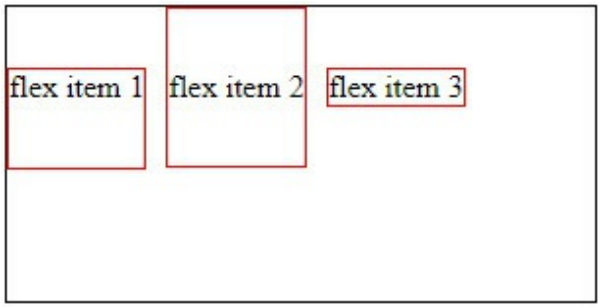


Example 4:

style.css

```
.flex-container {
  display: flex;
  align-items: baseline;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}

.item1 {
  height: 50px;
}
.item2 {
  line-height: 80px;
}
```

Result:



9.4.1.5 *flex-wrap*

The " flex-wrap " property specifies whether the flex items should wrap or not if there is not enough room for them on one flex line.

The possible values are as follows:

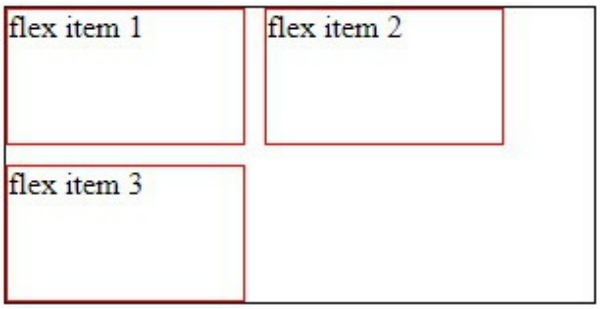
- nowrap (default value) - The flexible items will not wrap
- wrap - The flexible items will wrap if necessary
- wrap-reverse - The flexible items will wrap, if necessary, in reverse order

Example 1:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:

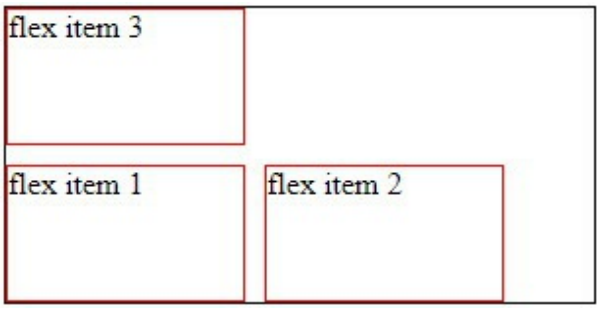


Example 2:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap-reverse;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:



9.4.1.6 align-content

The “ align-content ” property modifies the behavior of the “ flex-wrap ” property as it aligns flex lines of wrapping items. The possible values are as follows:

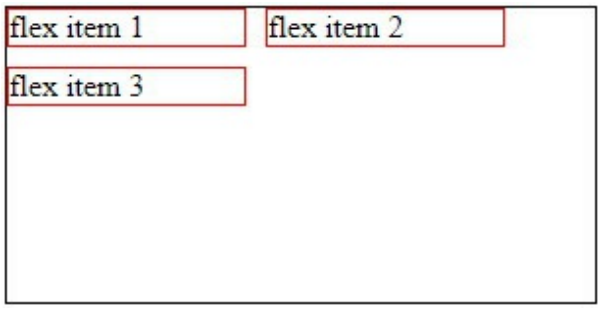
- stretch (default value) - Lines stretch to take up the remaining space
- flex-start - Lines are packed toward the start of the flex container
- flex-end - Lines are packed toward the end of the flex container
- center - Lines are packed toward the center of the flex container
- space-between - Lines are evenly distributed in the flex container
- space-around - Lines are evenly distributed in the flex container, with half-size spaces on either end
- space-evenly - Items will have equal space around them.

Example 1:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: flex-start;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:

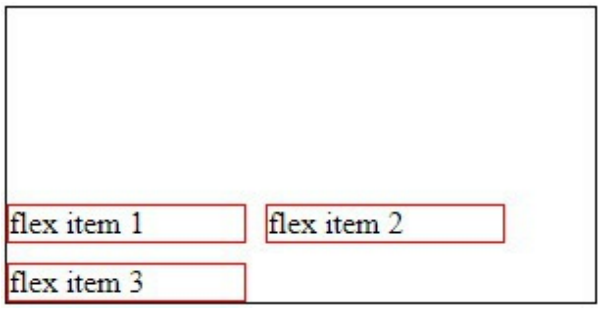


Example 2:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: flex-end;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:



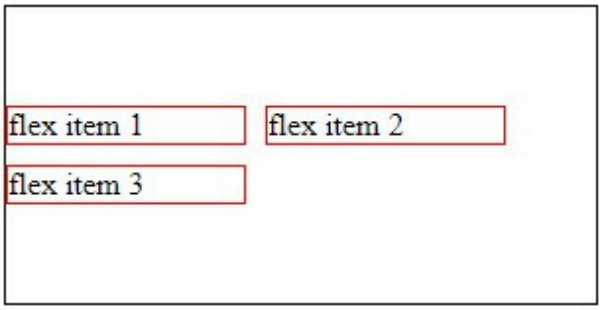
Example 3:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: center;
```

```
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:

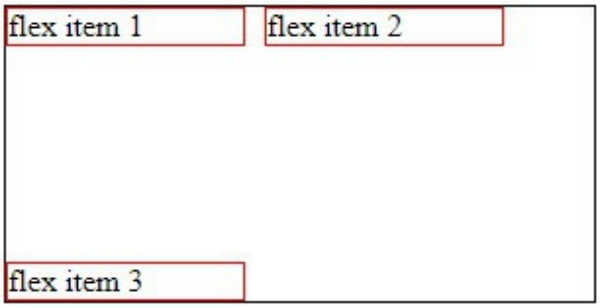


Example 4:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-between;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:

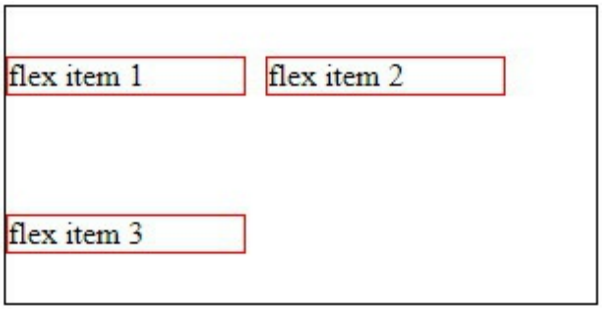


Example 5:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-around;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:



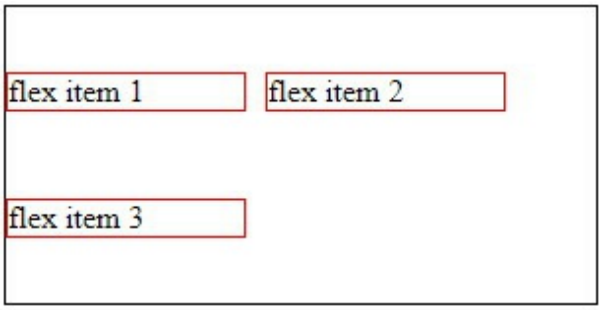
Example 6:

style.css

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-evenly;
```

```
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result:



).4.2 Flex Items

).4.2.1 flex-basis

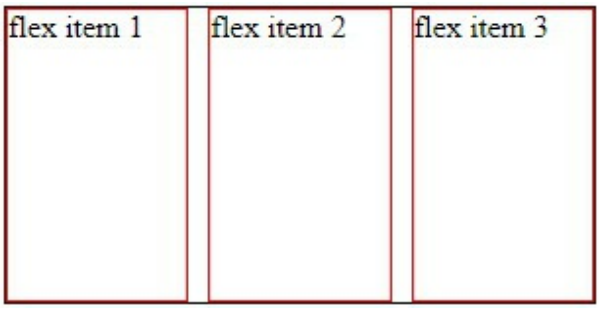
The “ flex-basis ” property defines an item’s width if the original width is smaller than the flex-basis value.

Example:

style.css

```
.flex-container {
  display: flex;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 120px;
}
```

Result: The flex items were expanded as much as possible to reach the flex-basis size.



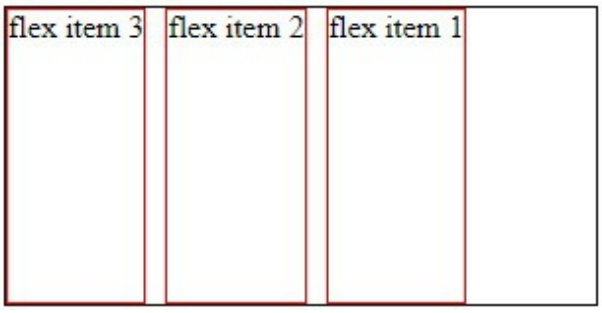
9.4.2.2 *order*

The “ order ” property controls the order of items. The items with no “ order ” value will be settled first; then come the smallest order value, and the highest order value makes the item last.

style.css

```
.flex-container {
  display: flex;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}

.item1 {
  order: 2;
}

.item2 {
  order: 1;
}
```

Result:



9.4.2.3 *flex-grow*

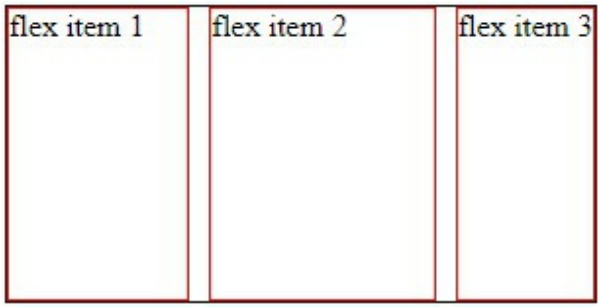
The “ flex-grow ” property allows an element to grow (0 means no).

Example:

style.css

```
.flex-container {
  display: flex;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}

.item1 {
  flex-grow: 1;
}
.item2 {
  flex-grow: 2;
}
```

Result: Item 2 gets double the available space compared to item 1.



9.4.2.4 *flex-shrink*

The “ flex-shrink ” property allows an element to shrink (0 means no).

Example:

index.html

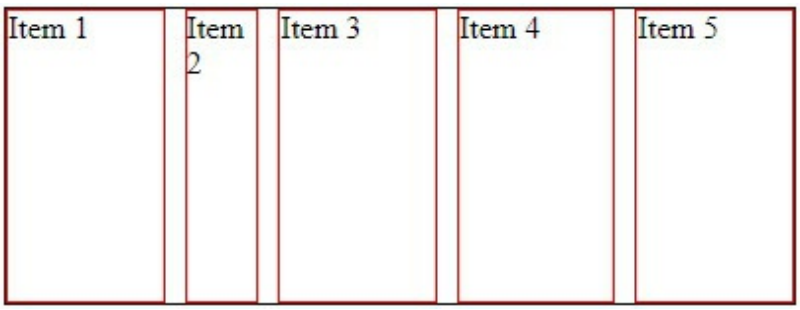
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
```

```
  <body>
    <div class="flex-container">
      <div class="flex-item item1">Item 1</div>
      <div class="flex-item item2">Item 2</div>
      <div class="flex-item item3">Item 3</div>
      <div class="flex-item item4">Item 4</div>
      <div class="flex-item item5">Item 5</div>
    </div>
  </body>
</html>
```

style.css

```
.flex-container {
  display: flex;
  width: 400px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
  flex-basis: 100px;
}

.item2 {
  flex-shrink: 3;
}
```

Result: Since the container width is 400px, but there are five items with flex-basis of 100px , all the items must be shrunk. The second item with " flex-shrink: 3 " will be the smallest one since its shrinking space is three times that of other items.



9.4.2.5 *align-self*

The “ align-self ” property of flex items overrides the “ align-items ” property of its parent.

Example:

index.html

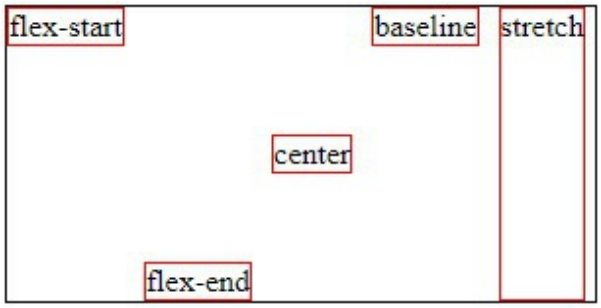
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item item1">flex-start</div>
      <div class="flex-item item2">flex-end</div>
      <div class="flex-item item3">center</div>
      <div class="flex-item item4">baseline</div>
      <div class="flex-item item5">stretch</div>
    </div>
  </body>
</html>
```

style.css

```
.flex-container {
  display: flex;
  align-items: center;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}

.item1 {
  align-self: flex-start;
}
.item2 {
  align-self: flex-end;
```

```
}

.item4 {
  align-self: baseline;
}
.item5 {
  align-self: stretch;
}
```

Result:



9.4.2.6 *flex*

The “ flex ” property is the shorthand for flex-grow , flex-shrink , and flex-basis .

```
flex: 0 1 auto;
```

In the following example, the first flex item will consume 2/4 of the free space, and the other two will consume 1/4 of the free space.

Example:

index.html

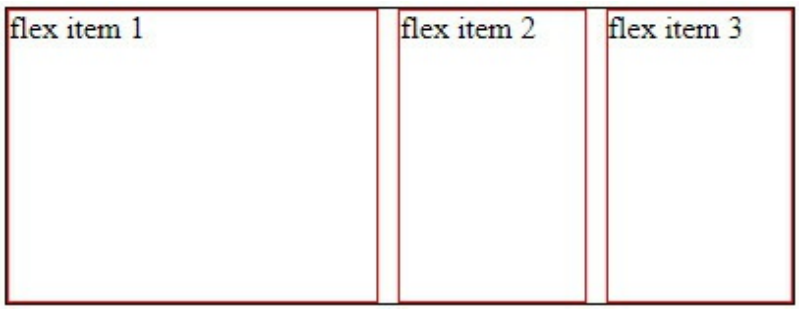
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item item1">flex item 1</div>
      <div class="flex-item item2">flex item 2</div>
      <div class="flex-item item3">flex item 3</div>
    </div>
```

```
  </body>
</html>
```

style.css

```
.flex-container {
  display: flex;
  width: 400px;
  height: 150px;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}

.item1 {
  flex: 2;
}
.item2 {
  flex: 1;
}
.item3 {
  flex: 1;
}
```

Result:



9.4.2.7 *Auto Margin for Flex Items*

We can implement our navigation bar like this example:

Example:

index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item lnk-home">HOME</div>
      <div class="flex-item">Contact</div>
      <div class="flex-item">Login</div>
    </div>
  </body>
</html>
```

style.css

```css
.flex-container {
  display: flex;
  width: 100%;
  border: 1px solid black;
  gap: 10px;
}

.flex-item {
  border: 1px solid red;
}

.lnk-home {
  margin-right: auto;
}
```

Result:

.5 CSS Grid

While CSS flexbox works with a 1D array of items, CSS grid provides a grid-based layout system with rows and columns. To define a grid container, we set the “ display ” property to “ grid ” to create a block-level grid or “ inline-grid ” to create an inline-level grid.

This file is used in the following examples:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="grid-container">
      <div class="grid-item item1">Item 1</div>
      <div class="grid-item item2">Item 2</div>
      <div class="grid-item item3">Item 3</div>
      <div class="grid-item item4">Item 4</div>
      <div class="grid-item item5">Item 5</div>
      <div class="grid-item item6">Item 6</div>
      <div class="grid-item item7">Item 7</div>
      <div class="grid-item item8">Item 8</div>
      <div class="grid-item item9">Item 9</div>
    </div>
  </body>
</html>
```

9.5.1 Grid Container

9.5.1.1 grid-template-columns

The “ grid-template-columns ” property defines the number of columns in the grid layout and the width of each column. The syntax is:

```
grid-template-columns: col-1-width col-2-width ... col-N-width
```

The column width can be:

- a fixed length like 60px
- auto - only take the necessary size to fill its content
- fr - flexible value, e.g., 1fr, 2fr , etc.

style.css

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6
Item 7	Item 8	Item 9

9.5.1.2 *grid-template-rows*

The “ grid-template-rows ” property defines the height of each row. The syntax is:

```
grid-template-rows: row-1-height row-2-height ... row-N-height
```

The row height can be:

- a fixed length like 60px
- auto - only take the necessary size to fill its content
- fr - flexible value, e.g., 1fr, 2fr , etc.

style.css

```
.grid-container {
  display: grid;
  grid-template-columns: 2fr 1fr 3fr;
  grid-template-rows: 30px auto 60px;
}

.grid-item {
  border: 1px solid red;
}
```

Result: The grid is horizontally divided into 6 fragments. Then, the first column gets two fragments, the second column gets one fragment, and the third column gets three fragments.

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6
Item 7	Item 8	Item 9

Instead of writing this CSS code:

```
grid-template-columns: 1fr 1fr 1fr;
```

we can write:

```
grid-template-columns: repeat(3, 1fr);
```

9.5.1.3 *column-gap*

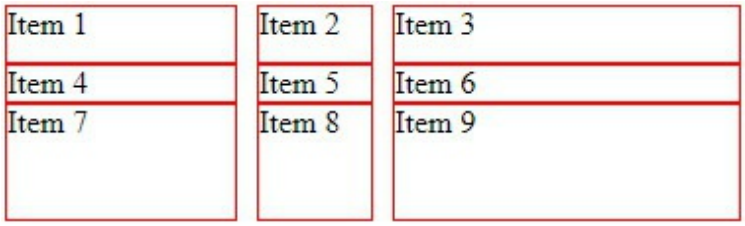
The “ column-gap ” property creates space between columns in the grid.

Example:

style.css

```
.grid-container {
  display: grid;
  grid-template-columns: 2fr 1fr 3fr;
  grid-template-rows: 30px auto 60px;
  column-gap: 10px;
}

.grid-item {
  border: 1px solid red;
}
```

Result:



9.5.1.4 *row-gap*

The “ row-gap ” property creates space between rows in the grid.

Example:

style.css

```
.grid-container {
  display: grid;
  grid-template-columns: 2fr 1fr 3fr;
  grid-template-rows: 30px auto 60px;
  row-gap: 10px;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

Item 1	Item 2	Item 3
Item 4	Item 5	Item 6
Item 7	Item 8	Item 9

9.5.1.5 *gap*

The “ gap ” property creates space between rows and columns in the grid. It is a shorthand for the “ row-gap ” and “ column-gap ” properties. The syntax is:

```
gap: row-gap column-gap
```

or if row-gap = column-gap = gap-size

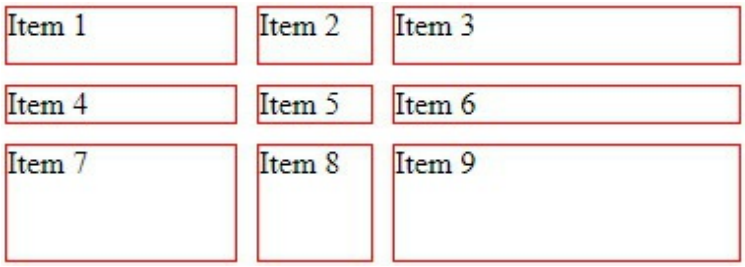
```
gap: gap-size
```

Example:

style.css

```
.grid-container {
  display: grid;
  grid-template-columns: 2fr 1fr 3fr;
  grid-template-rows: 30px auto 60px;
  gap: 10px;
}

.grid-item {
  border: 1px solid red;
}
```

Result:



9.5.1.6 *justify-items*

The “ justify-items ” property horizontally aligns the grid container's items when the items do not use all available space of the grid cells.

The possible values are as follows:

- stretch (default value) - Items are stretched to fit the cell
- start - Items are positioned on the left side of the cell
- end - Items are positioned on the right side of the cell
- center - Items are positioned in the center of the cell (horizontally)

Example 1:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-items: start;
}

.grid-item {
  border: 2px solid red;
}
```

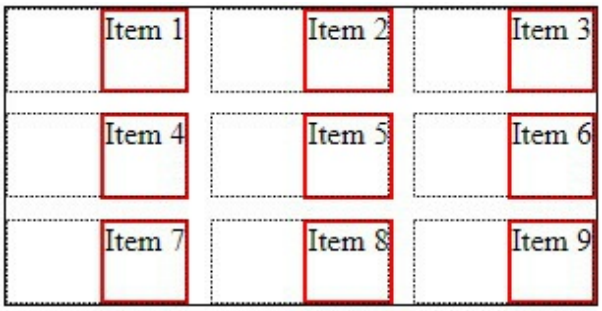
Result: (dotted lines are outlines of the grid cells)

Example 2:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-items: end;
}

.grid-item {
  border: 2px solid red;
}
```

Result: (dotted lines are outlines of the grid cells)

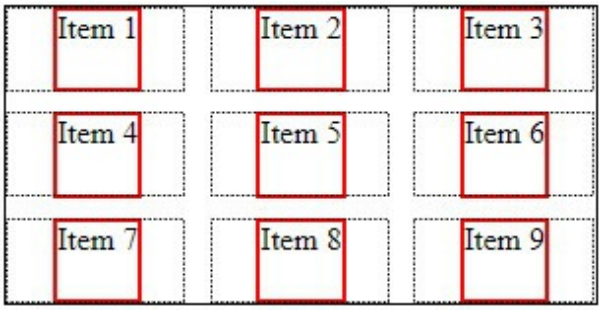


Example 3:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-items: center;
}

.grid-item {
  border: 2px solid red;
}
```

Result: (dotted lines are outlines of the grid cells)


Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9

9.5.1.7 *align-items*

The “ align-items ” property vertically aligns the grid container's items when the items do not use all available space of the grid cells.

The possible values are as follows:

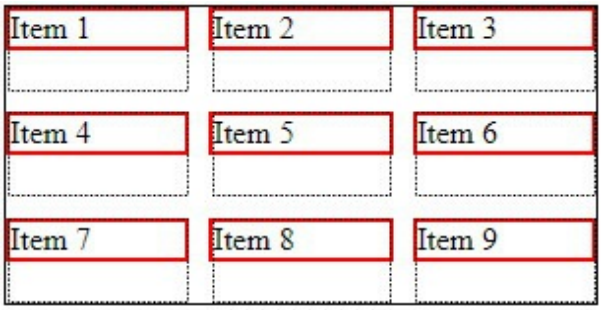
- stretch (default value) - Items are stretched to fit the cell
- start - Items are positioned at the top of the cell
- end - Items are positioned at the bottom of the cell
- center - Items are positioned in the center of the cell (vertically)
- baseline - Items are aligned along the text baseline of the cell

Example 1:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-items: start;
}

.grid-item {
  border: 2px solid red;
}
```

Result: (dotted lines are outlines of the grid cells)

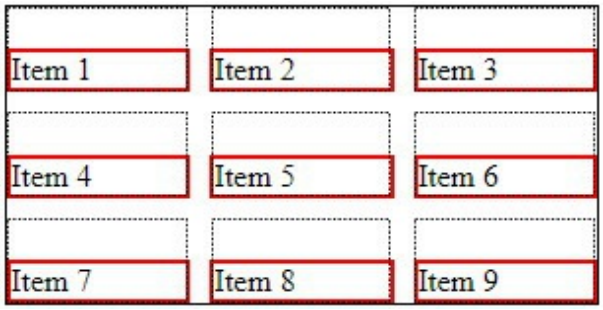


Example 2:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-items: end;
}

.grid-item {
  border: 2px solid red;
}
```

Result: (dotted lines are outlines of the grid cells)



Example 3:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-items: center;
}

.grid-item {
  border: 2px solid red;
}
```

Result: (dotted lines are outlines of the grid cells)


Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9

9.5.1.8 *justify-content*

The “ justify-content ” property horizontally aligns the grid container's items when the items do not use all available space.

The possible values are as follows:

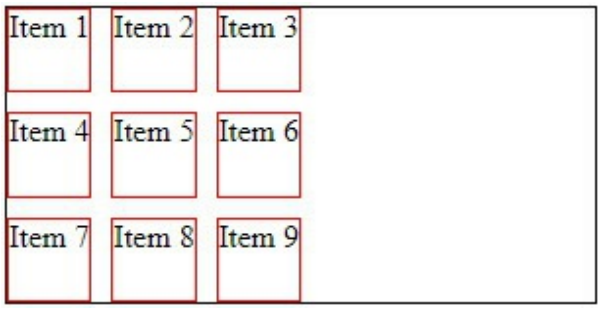
- start - Items are positioned at the beginning of the container.
- end - Items are positioned at the end of the container.
- center - Items are positioned at the center of the container.
- space-between - Items are positioned with space between the items.
- space-around - Items are positioned with space before, between, and after the items.
- space-evenly - Items will have equal space around them.

Example 1:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-content: start;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

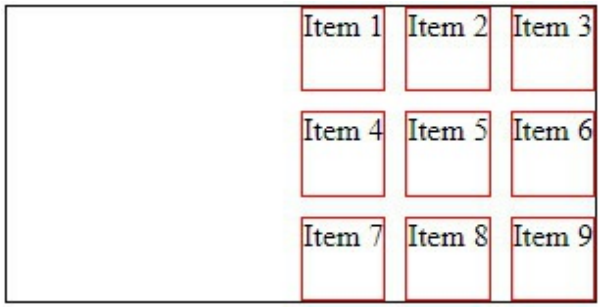


Example 2:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-content: end;
}

.grid-item {
  border: 1px solid red;
}
```

Result:



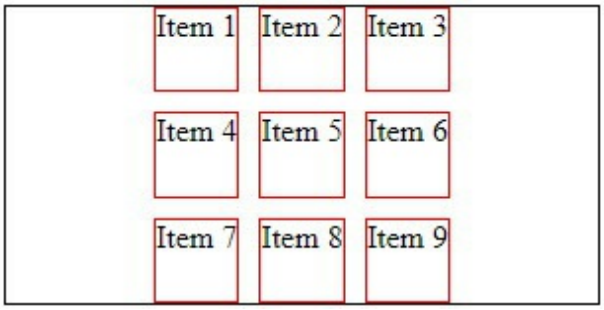
Example 3:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
```

```
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-content: center;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

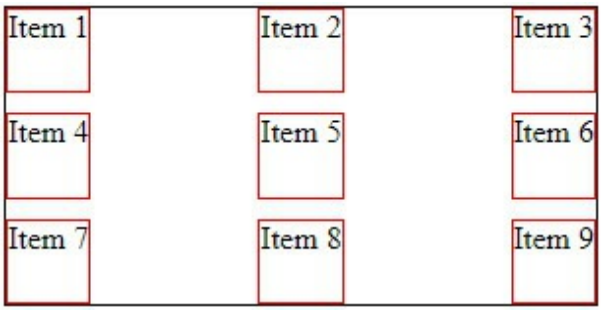


Example 4:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-content: space-between;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

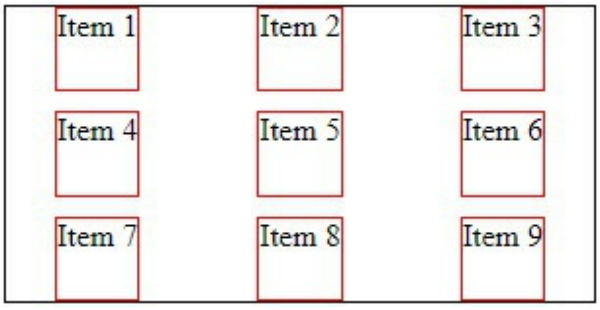


Example 5:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-content: space-around;
}

.grid-item {
  border: 1px solid red;
}
```

Result:



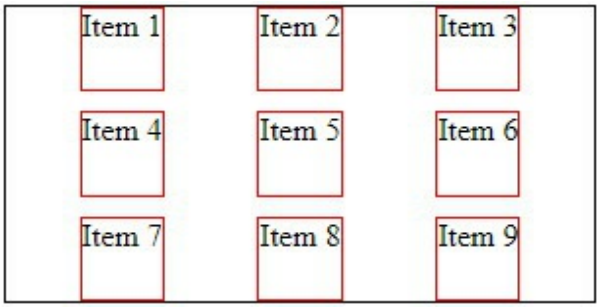
Example 6:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
```

```
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-content: space-evenly;
}

.grid-item {
  border: 1px solid red;
}
```

Result:



9.5.1.9 *align-content*

The “ align-content ” property vertically aligns the rows inside the grid’s container. The possible values are as follows:

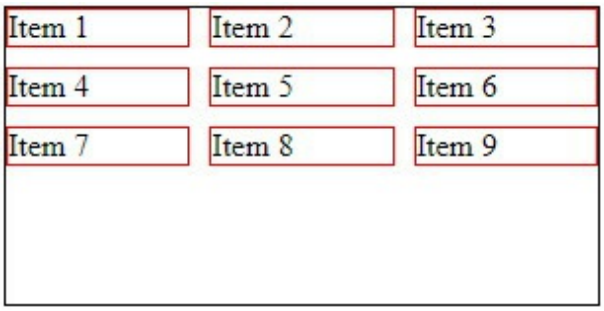
- start - Rows are packed toward the top of the grid container
- end - Rows are packed toward the bottom of the grid container
- center - Rows are packed toward the center of the grid container
- space-between - Rows are evenly distributed in the grid container
- space-around - Rows are evenly distributed in the grid container, with half-size spaces on either end
- space-evenly - Items will have equal space around them.

Example 1:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-content: start;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

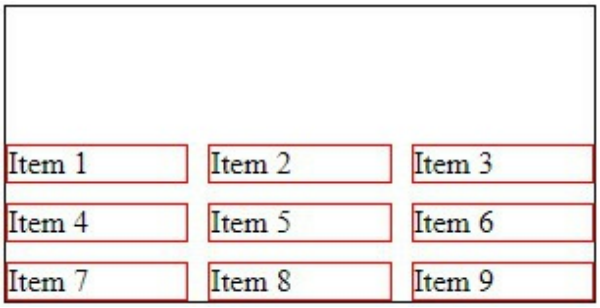


Example 2:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-content: end;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

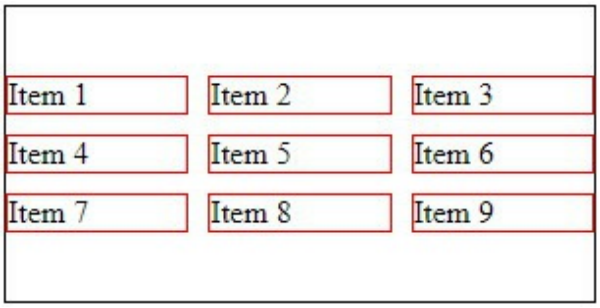


Example 3:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-content: center;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

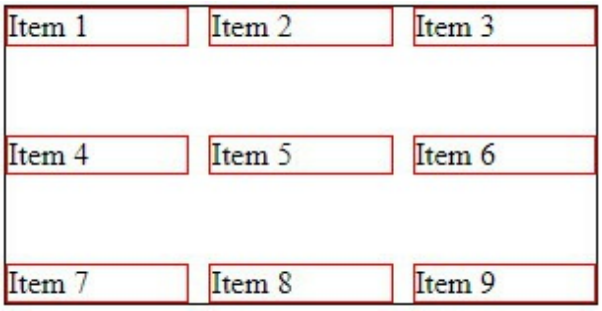


Example 4:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-content: space-between;
}

.grid-item {
  border: 1px solid red;
}
```

Result:



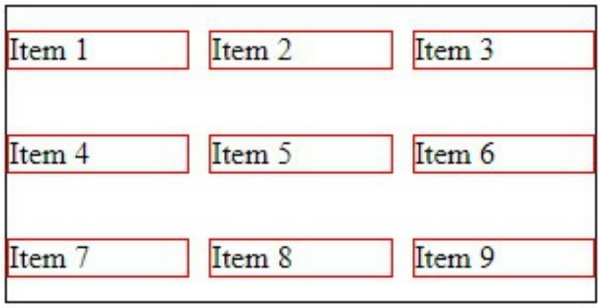
Example 5:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
```

```
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-content: space-around;
}

.grid-item {
  border: 1px solid red;
}
```

Result:

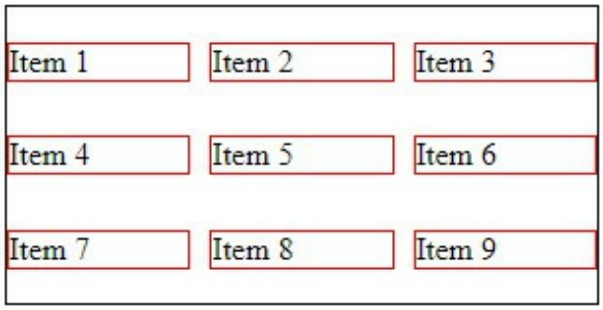


Example 6:

style.css

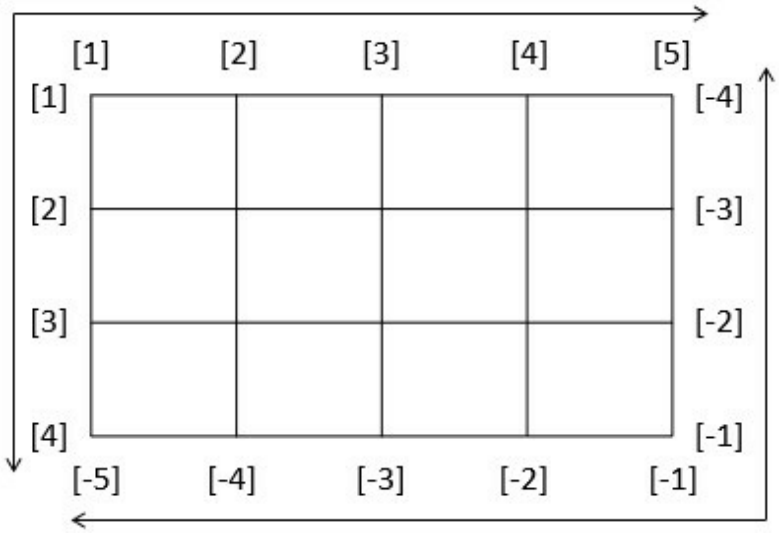
```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-content: space-evenly;
}

.grid-item {
  border: 1px solid red;
}
```

Result:


Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9

9.5.2 Grid Items

We use grid lines to specify the location of a grid item. The grid lines are assigned numbers along the X-axis and Y-axis. Horizontal lines are called row-lines, and vertical lines are called column-lines.



A grid line can be assigned a positive number or a negative number. For example, in the above grid, the second row-line can be 2 or -3; and the second column-line can be 2 or -4.

9.5.2.1 grid-column-start

The " grid-column-start " property defines a grid item's start location within the grid by referring to the grid column-lines. The syntax is:

```
grid-column-start: start-column
```

start-column could have these values:

- <line number> - a number refers to a numbered grid line.
- span <number> - the item will span the provided number of cells.

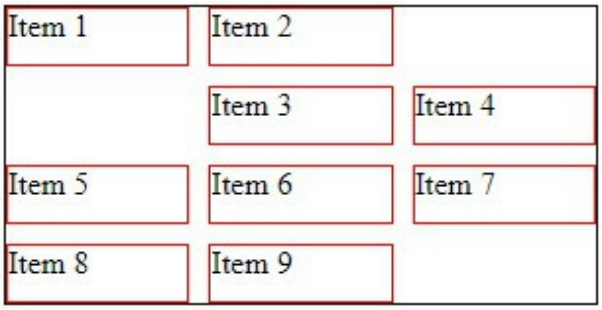
Example:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
```

```
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
}

.grid-item {
  border: 1px solid red;
}

.item3 {
  grid-column-start: 2;
}
```

Result:



9.5.2.2 *grid-column-end*

The " grid-column-end " property is similar to the " grid-column-start " property, but it defines the ending location.

9.5.2.3 *grid-column*

The " grid-column " property is a shorthand for the " grid-column-start " and " grid-column-end " properties. The syntax is:

```
grid-column: start-column / end-column
```

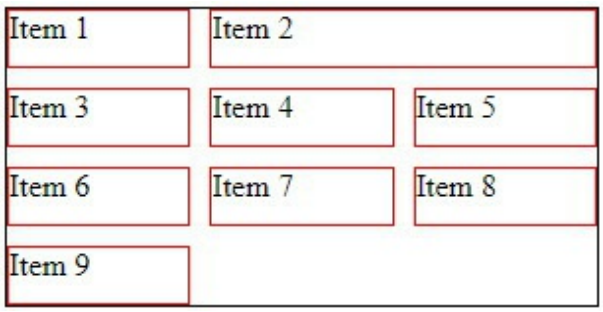
Example:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
```

```
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
}

.grid-item {
  border: 1px solid red;
}

.item2 {
  /* grid-column: 2 / 4; */
  /* grid-column: 2 / -1; */
  grid-column: 2 / span 2;
}
```

Result:



9.5.2.4 *grid-row-start*

The " grid-row-start " property defines a grid item's start location within the grid by referring to the grid row-lines. The syntax is:

```
grid-row-start: start-row
```

start-row could have these values:

- <line number> - a number refers to a numbered grid line.
- span <number> - the item will span the provided number of cells.

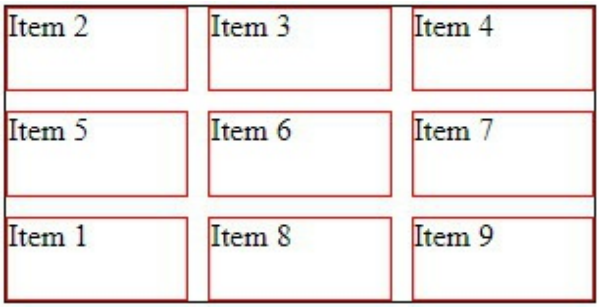
Example:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
```

```
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
}

.grid-item {
  border: 1px solid red;
}

.item1 {
  grid-row-start: 3;
}
```

Result:



9.5.2.5 *grid-row-end*

The “ grid-row-end ” property is similar to the “ grid-row-start ” property, but it defines the ending location.

9.5.2.6 *grid-row*

The “ grid-row ” property is a shorthand for the “ grid-row-start ” and “ grid-row-end ” properties. The syntax is:

```
grid-row: start-row / end-row
```

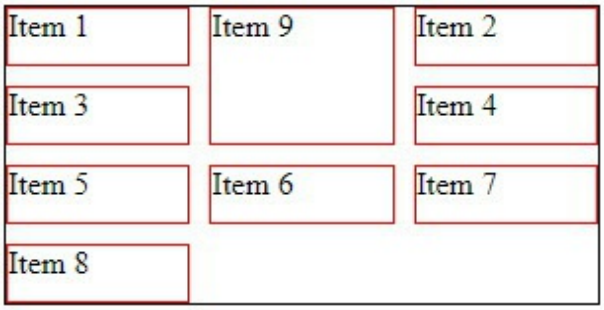
Example:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
```

```
  grid-template-columns: auto auto auto;
  gap: 10px;
}

.grid-item {
  border: 1px solid red;
}

.item9 {
  grid-column-start: 2;
  /* grid-row: 1 / 3; */
  grid-row: 1 / span 2;
}
```

Result:



9.5.2.7 *grid-area*

The “ grid-area ” property is a shorthand property for the “ grid-row-start ,” “ grid-column-start ,” “ grid-row-end ,” and the “ grid-column-end ” properties.

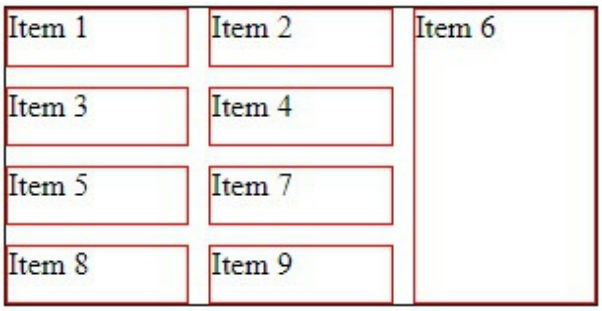
Example:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
}

.grid-item {
```

```
  border: 1px solid red;
}

.item6 {
  grid-area: 1 / 3 / 5 / 4;
}
```

Result: “item6” starts on row-line 1 and column-line 3 and ends on row-line 5 and column-line 4.



9.5.2.8 *justify-self*

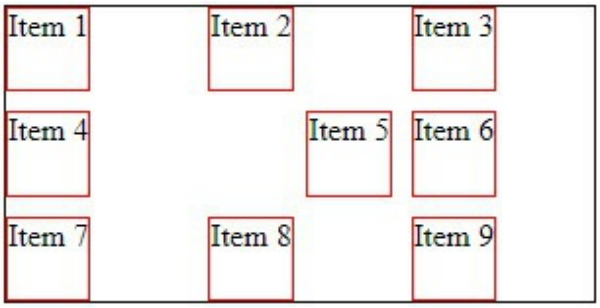
The “ justify-self ” property overrides the “ justify-items ” property of the grid container.

Example:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-items: start;
}

.grid-item {
  border: 1px solid red;
}

.item5 {
  justify-self: end;
}
```

Result:



9.5.2.9 *align-self*

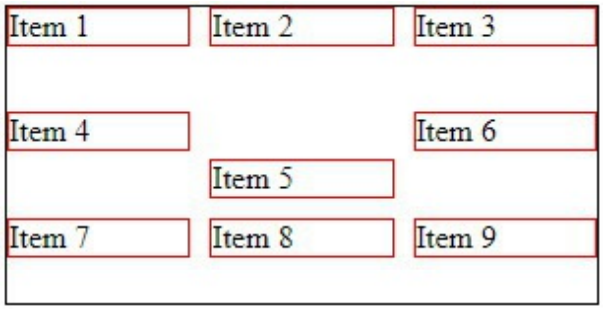
The “ align-self ” property overrides the “ align-items ” property of the grid container.

Example:

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
  align-items: start;
}

.grid-item {
  border: 1px solid red;
}

.item5 {
  align-self: end;
}
```

Result:



9.5.2.10 *grid-template-areas*

The “ grid-template-areas ” property specifies areas within the grid layout. We name the grid items using the “ grid-area ” property and then refer to the name in the “ grid-template-areas ” property.

Example:

index.html

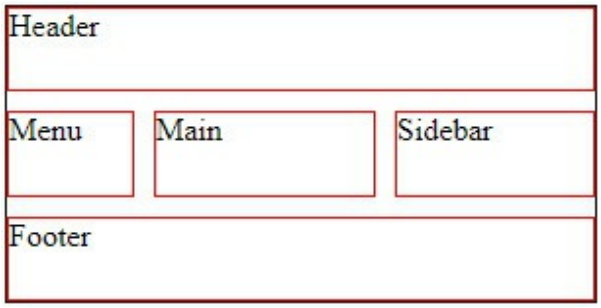
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="grid-container">
      <div class="grid-item item1">Header</div>
      <div class="grid-item item2">Menu</div>
      <div class="grid-item item3">Main</div>
      <div class="grid-item item4">Sidebar</div>
      <div class="grid-item item5">Footer</div>
    </div>
  </body>
</html>
```

style.css

```
.grid-container {
  display: grid;
  width: 300px;
  height: 150px;
  border: 1px solid black;
  grid-template-columns: auto auto auto;
  gap: 10px;
```

```
  grid-template-areas:
    "header header header header header header"
    "menu main main main sidebar sidebar"
    "footer footer footer footer footer footer";
}

.grid-item {
  border: 1px solid red;
}

.item1 {
  grid-area: header;
}
.item2 {
  grid-area: menu;
}
.item3 {
  grid-area: main;
}
.item4 {
  grid-area: sidebar;
}
.item5 {
  grid-area: footer;
}
```

Result:



.6 CSS Alignment

9.6.1 Horizontal Alignment

We use the CSS “ text-align ” property to align text. Please refer to the section “CSS Text” for more details.

To align a block element center inside its containing element, we use the CSS “ margin: auto; .”

Example:

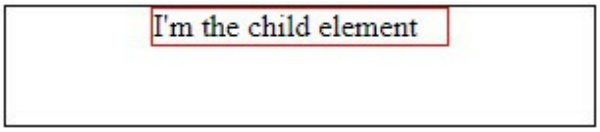
index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <div class="container">
   <div class="child">I'm the child element</div>
  </div>
 </body>
</html>
```

style.css

```
.container {
 border: 1px solid black;
 width: 300px;
 height: 60px;
}

.child {
 border: 1px solid red;
 width: 150px;
 margin: auto;
}
```

Result:



Therefore, to center align an image, we could change it to a block element and use “ margin: auto; ” like this.

```
img {
  display: block;
  margin: auto;
}
```

We can use “ position: absolute; ” or “ float ” to align elements left or right. Please refer to sections “CSS Position” and “CSS Float” sections in the chapter “CSS Layout” for more details.

.6.2 Vertical alignment

We can use “ margin-top ” and “ margin-bottom ” properties to make an element center vertically inside its containing element.

Example:

index.html

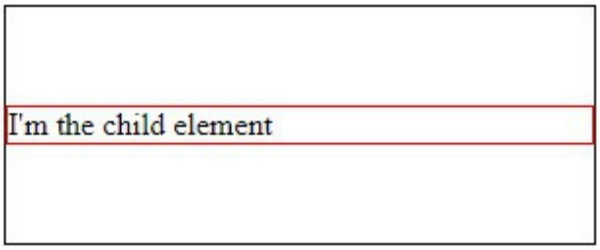
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div class="child">I'm the child element</div>
    </div>
  </body>
</html>
```

style.css

```
.container {
  border: 1px solid black;
  width: 300px;
}
```

```
.child {
  border: 1px solid red;
  margin-top: 50px;
  margin-bottom: 50px;
}
```

Result:



Or we can use the CSS line-height property as in the chapter "**CSS Text**" to vertically center align the text.

9.6.3 Perfect Centering

9.6.3.1 Centering Using The position And transform Properties

The CSS “ margin ” and “ line-height ” properties only work if the containing element has only one child. Therefore, in some cases as below example, we need a better way to vertically center align an element using CSS “ position ” and “ transform ” properties.

Example:

index.html

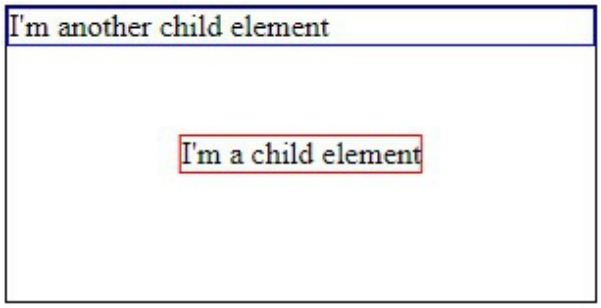
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <div class="child">I'm a child element</div>
      <div class="another-child">I'm another child element</div>
    </div>
  </body>
</html>
```

style.css

```
.container {
  position: relative;
  border: 1px solid black;
  width: 300px;
  height: 150px;
}

.child {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  border: 1px solid red;
}
```

```
.another-child {
  border: 1px solid blue;
}
```

Result:



9.6.3.2 *Centering Using The flex Property*

Example:

index.html

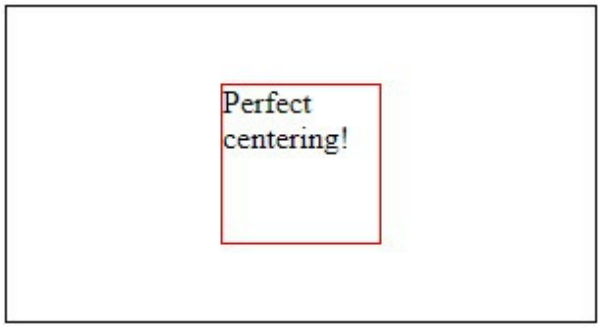
```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="flex-container">
      <div class="flex-item">Perfect centering!</div>
    </div>
  </body>
</html>
```

style.css

```
.flex-container {
  display: flex;
  width: 300px;
  height: 160px;
  border: 1px solid black;
}

.flex-item {
  border: 1px solid red;
```

```
  width: 80px;
  height: 80px;
  margin: auto;
}
```

Result:



CHAPTER 20
CSS Transitions and Animations

.1 CSS Transitions

CSS transitions allow you to change CSS property values smoothly (from one value to another) over a given duration.

We will use this file in the following examples:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>Hover the mouse on this element to see the effect!</div>
  </body>
</html>
```

.1.1 Basic Transitions

To create a transition effect, you must specify two things:

- the CSS property you want to add an effect to
- the duration of the effect

Example 1:

style.css

```
div {
  width: 100px;
  height: 100px;
  color: white;
  background: red;
  transition: width 2s;
}

div:hover {
  width: 300px;
}
```

Example 2:

style.css

```
div {
  width: 100px;
  height: 100px;
  color: white;
  background: red;
  transition: width 2s, height 4s, transform 2s;
}

div:hover {
  width: 300px;
  height: 300px;
  transform: skewX(50deg);
}
```

.1.2 Transition Delay

The “ transition-delay ” property specifies a delay (in seconds) for the

transition effect.

Example:

style.css

```
div {
  width: 100px;
  height: 100px;
  color: white;
  background: red;
  transition: width 2s, height 4s;
  transition-delay: 1s;
}

div:hover {
  width: 300px;
  height: 300px;
}
```

.1.3 Transition Timing Function

The " transition-timing-function " property specifies the speed curve of the transition effect. It can have the following values:

- ease - specifies a transition effect with a slow start, then fast, then end slowly (this is the default)
- linear - specifies a transition effect with the same speed from start to end
- ease-in - specifies a transition effect with a slow start
- ease-out - specifies a transition effect with a slow end
- ease-in-out - specifies a transition effect with a slow start and end

Example:

style.css

```
div {
  width: 100px;
  height: 100px;
  color: white;
  background: red;
  transition: width 2s, height 4s;
```

```
  transition-timing-function: ease-out;
}

div:hover {
  width: 300px;
  height: 300px;
}
```

.1.4 Shorthand Transition Property

The CSS transition properties can be specified one by one, like this:

```
div {
  transition-property: width;
  transition-duration: 2s;
  transition-timing-function: linear;
  transition-delay: 1s;
}
```

or by using the shorthand property “ transition ”:

```
div {
  transition: width 2s linear 1s;
}
```

.2 CSS Animations

Animations let HTML elements gradually change from one style to another. To use CSS animation, you must first define some “ key-frames ” for the animation. Key-frames hold what styles the element will have at certain times.

We will use this file in the following examples:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div></div>
  </body>
</html>
```

.2.1 Two-step Keyframes

The following example binds the " my-animation " animation to the element. The animation will last for 4 seconds, and it will gradually change the background-color of the element from " red " to " yellow ":

Example:

style.css

```
@keyframes my-animation {
  from {
    background-color: red;
  }
  to {
    background-color: yellow;
  }
}

div {
  width: 100px;
  height: 100px;
  background-color: green;
  animation-name: my-animation;
  animation-duration: 4s;
}
```

.2.2 Multiple Keyframes

It is also possible to use percentages from 0% to 100% like this.

Example:

style.css

```
@keyframes my-animation {
  0% {
    background-color: red;
  }
  25% {
    background-color: yellow;
  }
  50% {
    background-color: blue;
  }
  100% {
    background-color: green;
  }
}

div {
  width: 100px;
  height: 100px;
  background-color: cyan;
  animation-name: my-animation;
  animation-duration: 4s;
}
```

.2.3 Animation Delay

The “ animation-delay ” property specifies a delay for the start of an animation.

Example:

style.css

```
@keyframes my-animation {
  0% {
    background-color: red;
  }
  25% {
    background-color: yellow;
  }
  50% {
    background-color: blue;
  }
  100% {
    background-color: green;
  }
}

div {
  width: 100px;
  height: 100px;
  background-color: cyan;
  animation-name: my-animation;
  animation-duration: 4s;
  animation-delay: 2s;
}
```

.2.4 Animation Iteration Count

The “ animation-iteration-count ” property specifies the number of times an animation should run. We can set its value to a number or " infinite " (the animation continue forever).

Example:

style.css

```
@keyframes my-animation {
  0% {
    background-color: red;
  }
  25% {
    background-color: yellow;
  }
  50% {
    background-color: blue;
  }
  100% {
    background-color: green;
  }
}

div {
  width: 100px;
  height: 100px;
  background-color: cyan;
  animation-name: my-animation;
  animation-duration: 4s;
  animation-iteration-count: 3;
}
```

.2.5 Animation Direction

The “ animation-direction ” property lets an animation run in reverse direction or alternate cycles.

Example 1:

style.css

```
@keyframes my-animation {
  from {
    left: 0;
  }
  to {
    left: 300px;
  }
}

div {
  position: relative;
  width: 100px;
  height: 100px;
  background-color: cyan;
  animation-name: my-animation;
  animation-duration: 4s;
  animation-iteration-count: 3;
  animation-direction: reverse;
}
```

Result: The box is moved from the right (left: 300px) to the left (left: 0) three times.

Example 2:

style.css

```
@keyframes my-animation {
  from {
    left: 0;
  }
  to {
    left: 300px;
  }
}
```

```
div {
  position: relative;
  width: 100px;
  height: 100px;
  background-color: cyan;
  animation-name: my-animation;
  animation-duration: 4s;
  animation-iteration-count: 3;
  animation-direction: alternate;
}
```

Result: The box is moved three times

- from the left (left: 0) to the right (left: 300px)
- from the right (left: 300px) to the left (left: 0)
- from the left (left: 0) to the right (left: 300px)

.2.6 Animation Timing Function

The “ animation-timing-function ” property specifies the speed curve of the animation. It can have the following values:

- ease - specifies an animation with a slow start, then fast, then end slowly (this is the default)
- linear - specifies an animation with the same speed from start to end
- ease-in - specifies an animation with a slow start
- ease-out - specifies an animation with a slow end
- ease-in-out - specifies an animation with a slow start and end

Example:

style.css

```
@keyframes my-animation {
  from {
    left: 0;
  }
  to {
    left: 300px;
  }
}

div {
  position: relative;
  width: 100px;
  height: 100px;
  background-color: cyan;
  animation-name: my-animation;
  animation-duration: 4s;
  animation-timing-function: ease-out;
}
```

2.7 Shorthand Animation Property

The “ animation ” property is the shorthand of the above animation properties. For example:

style.css

```
div {
  position: relative;
  width: 100px;
  height: 100px;
  background-color: cyan;

  animation-name: my-animation;
  animation-duration: 4s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
  animation-timing-function: linear;
}
```

can be written like this:

```
div {
  position: relative;
  width: 100px;
  height: 100px;
  background-color: cyan;

  animation: my-animation 4s linear infinite alternate;
}
```

CHAPTER 21
Advanced CSS

.1 CSS Variables

CSS variables help us:

- makes our CSS code much easier to maintain since we only change the value of one variable, and it will affect wherever the variable used
- makes the code more readable since we use the names of the variables

We use the var() function to get the value of a CSS variable. Moreover, variables are only available within the scope of the CSS selector that defines them. The syntax is:

```
selector {
  --your-variable-name: the-value;
}
```

Example:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <section>
      <header>
        <h2>This is a heading</h2>
      </header>
      <p>This is a paragraph.</p>
      <footer>
        <p>This is a footer.</p>
      </footer>
```

```
  </section>
  <footer>
    <p>This is the main footer.</p>
  </footer>
 </body>
</html>
```

style.css

```
:root {
 --a-global-color: red;
}

section {
 --a-local-color: blue;
}

header {
 color: var(--a-global-color);
}

footer {
 color: var(--a-local-color);
}
```

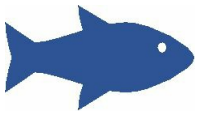
Result: The main footer is not a child of the <section> element, so it cannot access the " --a-local-color " variable.

This is a heading

This is a paragraph.

This is a footer.

This is the main footer.



In HTML, the :root selector always matches the <html> element.

.2 CSS Calculation

The calc() function is used to calculate the property value in CSS dynamically. Basic math operators like addition (+), subtraction (-), division (/), multiplication (*) and parentheses are allowed in calc() . For example:

```
width: calc((100%/3  -  50px) + (1em*2));
```

Example:

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <header>Header</header>
  <div class="main-content">
   <p>This is a paragraph</p>
   <p class="long-content">This is a long paragraph</p>
  </div>
  <footer>Footer</footer>
 </body>
</html>
```

style.css

```
header,
footer {
 height: 30px;
 line-height: 30px;
}

header {
 border-bottom: 1px solid black;
}
footer {
 border-top: 1px solid black;
 text-align: center;
```

```
}

.main-content {
  min-height: calc(100vh - 96px);
}

.long-content {
  border: 1px solid #ccc;
  /* height: 1000px; */
}
```

Open this web page in a browser and try to change the height of the browser; you will see that <footer> is always at the bottom of the page.

Then, uncomment the CSS “ height ” property of the <p class="long-content"> element to see that the code still works well with long content.

.3 CSS Counters

The CSS counter helps us create automatic numbering outlines without any JavaScript code.

- counter-reset - Creates a counter by giving it a name
- counter-increment - Increments a counter value
- content - Inserts generated content
- counter() function - Adds the value of a counter to an element

Example:

index.html

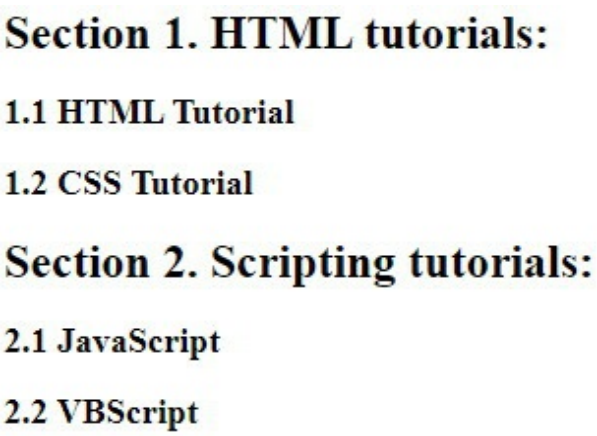
```
<!DOCTYPE html>
<html>
 <head>
  <title>My Website</title>
  <link rel="stylesheet" href="style.css" />
 </head>
 <body>
  <h1>HTML tutorials:</h1>
  <h2>HTML Tutorial</h2>
  <h2>CSS Tutorial</h2>

  <h1>Scripting tutorials:</h1>
  <h2>JavaScript</h2>
  <h2>VBScript</h2>
 </body>
</html>
```

style.css

```
body {
 font-size: 12px;
 counter-reset: my-section;
}

h1 {
 counter-reset: my-sub-section;
}

h1::before {
```

```
  counter-increment: my-section;
  content: "Section " counter(my-section) ". ";
}

h2::before {
  counter-increment: my-sub-section;
  content: counter(my-section) "." counter(my-sub-section) " ";
}
```

Result:



CHAPTER 22
Responsive CSS

.1 Responsive Web Design

Web pages can be viewed using many devices: desktops, tablets, phones, etc. Your web page should look good and be easy to use, regardless of the device. Responsive web design (RWD) can be done using only HTML and CSS.

.2 RWD Viewport

The viewport is the user's visible area of a web page. To configure the viewport correctly, you should include the following <meta> element in all your web pages.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

The “ width=device-width ” part sets the page’s width to follow the device’s screen width (which will vary depending on the device).

The “ initial-scale=1.0 ” part sets the initial zoom level when the browser loads the page.

Below are two screenshots of a web page on a mobile device without and with the viewport meta tag.

.3 CSS Media

.3.1 Media Types

The @media is used to define different style rules for different media types. You could have one set of style rules for computer screens, one for printers, one for handheld devices, one for television-type devices, and so on.

- all - Used for all media type devices (default)
- print - Used for printers
- screen - Used for computer screens, tablets, smartphones, etc.

Multiple comma-separated media types can be declared in the attribute. For example:

```
media="screen,print"
```

.3.2 Media Queries

Instead of looking for a type of device, they look at the capability of the device:

- width and height of the viewport
- width and height of the device
- orientation (Is the tablet/phone in landscape or portrait mode?)
- resolution

A media query consists of a media type and can contain one or more expressions that resolve to either true or false. The syntax is:

```
@media not|only mediatype and (expressions) {
  CSS-code;
}
```

in the <head> element:

```
<link rel="stylesheet" media="mediatype and|not|only (expressions)" href="style.css">
```

In the expression, these features are popularly used:

- aspect-ratio - The ratio between the width and the height of the viewport
- height - The viewport height
- max-aspect-ratio - The maximum ratio between the width and the height of the display area
- max-height - The maximum height of the display area, such as a browser window
- max-resolution - The maximum resolution of the device, using dpi or dpcm
- max-width - The maximum width of the display area, such as a browser window
- min-aspect-ratio - The minimum ratio between the width and the height of the display area
- min-height - The minimum height of the display area, such as a browser window
- min-resolution - The minimum resolution of the device, using dpi or dpcm
- min-width - The minimum width of the display area, such as a browser window
- orientation - The orientation of the viewport (landscape or portrait mode)
- resolution - The resolution of the output device, using dpi or dpcm
- width - The viewport width

The following example changes the background-color to green if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the background-color will be blue).

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

@media screen and (min-width: 480px) {
  body {
```

```
    background-color: green;
  }
}
```

Media queries can also be used to change a page's layout depending on the browser's orientation.

```
@media only screen and (orientation: landscape) {
  body {
    background-color: grey;
  }
}
```

.4 RWD Grid View

Many web pages are based on a grid view, which means that the page is divided into columns. A responsive grid view often has 12 columns and a total width of 100%. The columns will shrink and expand as you resize the browser window.

Firstly, ensure that all HTML elements have the box-sizing property set to border-box . It ensures that the padding and border are included in the total width and height of the elements.

```
* {
  box-sizing: border-box;
}
```

Then, we must calculate the percentage for one column: 100% / 12 columns = 8.33%.

After that, we make one class for each of the 12 columns, class="col-" and a number defining how many columns the section should span:

```
.col-1 { width: 8.33%; }
.col-2 { width: 16.66%; }
.col-3 { width: 25%; }
.col-4 { width: 33.33%; }
.col-5 { width: 41.66%; }
.col-6 { width: 50%; }
.col-7 { width: 58.33%; }
.col-8 { width: 66.66%; }
.col-9 { width: 75%; }
.col-10 { width: 83.33%; }
.col-11 { width: 91.66%; }
.col-12 { width: 100%; }
```

All these columns have to be floating to the left.

```
[class*="col-"] {
  float: left;
}
```

Each row should be wrapped in a . The number of columns inside

a row should always add up to 12x.

```
<div class="row">
  <div class="col-3">...</div>
  <div class="col-9">...</div>
</div>
```

The columns inside a row are all floating to the left. Therefore, when a row does not use all its columns, other elements behind it or from the next row could be placed to fill that unused space. To prevent this, we will add a style that clears the flow.

```
.row::after {
  content: "";
  clear: both;
  display: block;
}
```

The style automatically adds an empty (height = 0px) block-element right after each row and sets its style to clear the floating (clear: both).

We will add breakpoints to look good on many screen sizes using @media. And we will design for a smaller screen size first. It will make the page display faster on smaller devices, called mobile-first responsive CSS.

rwd-grid.css

```
* {
  box-sizing: border-box;
}

.row::after {
  content: "";
  clear: both;
  display: block;
}

/* For mobile phones: */
[class*="col-"] {
  float: left;
  width: 100%;
}

@media only screen and (min-width: 600px) {
```

```
  /* For tablets: */
  .col-m-1 { width: 8.33%; }
  .col-m-2 { width: 16.66%; }
  .col-m-3 { width: 25%; }
  .col-m-4 { width: 33.33%; }
  .col-m-5 { width: 41.66%; }
  .col-m-6 { width: 50%; }
  .col-m-7 { width: 58.33%; }
  .col-m-8 { width: 66.66%; }
  .col-m-9 { width: 75%; }
  .col-m-10 { width: 83.33%; }
  .col-m-11 { width: 91.66%; }
  .col-m-12 { width: 100%; }
}

@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 { width: 8.33%; }
  .col-2 { width: 16.66%; }
  .col-3 { width: 25%; }
  .col-4 { width: 33.33%; }
  .col-5 { width: 41.66%; }
  .col-6 { width: 50%; }
  .col-7 { width: 58.33%; }
  .col-8 { width: 66.66%; }
  .col-9 { width: 75%; }
  .col-10 { width: 83.33%; }
  .col-11 { width: 91.66%; }
  .col-12 { width: 100%; }
}
```

style.css

```
.row div {
  padding: 8px;
  border: 1px solid black;
}
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Website</title>
    <link rel="stylesheet" href="rwd-grid.css" />
```

```
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="row">
      <div class="col-3 col-m-3">DIV 1</div>
      <div class="col-6 col-m-9">DIV 2</div>
      <div class="col-3 col-m-12">DIV 3</div>
    </div>
  </body>
</html>
```

Result:

For mobiles having a screen width smaller than 600px, the style [class*="col-"] will be applied.

1	2	3	4	5	6	7	8	9	10	11	12
DIV 1											
DIV 2											
DIV 3											

For tablets having screen widths between 600px and 768px, styles of col-m-3, col-m-9 and col-m-12 will be applied since they override the style [class*="col-"] .

1	2	3	4	5	6	7	8	9	10	11	12
DIV 1			DIV 2								
DIV 3											

For desktops with a screen width larger than 768px, col-3 and col-6 will be applied since they override the style [class*="col-"] .

1	2	3	4	5	6	7	8	9	10	11	12
DIV 1			DIV 2						DIV 3		

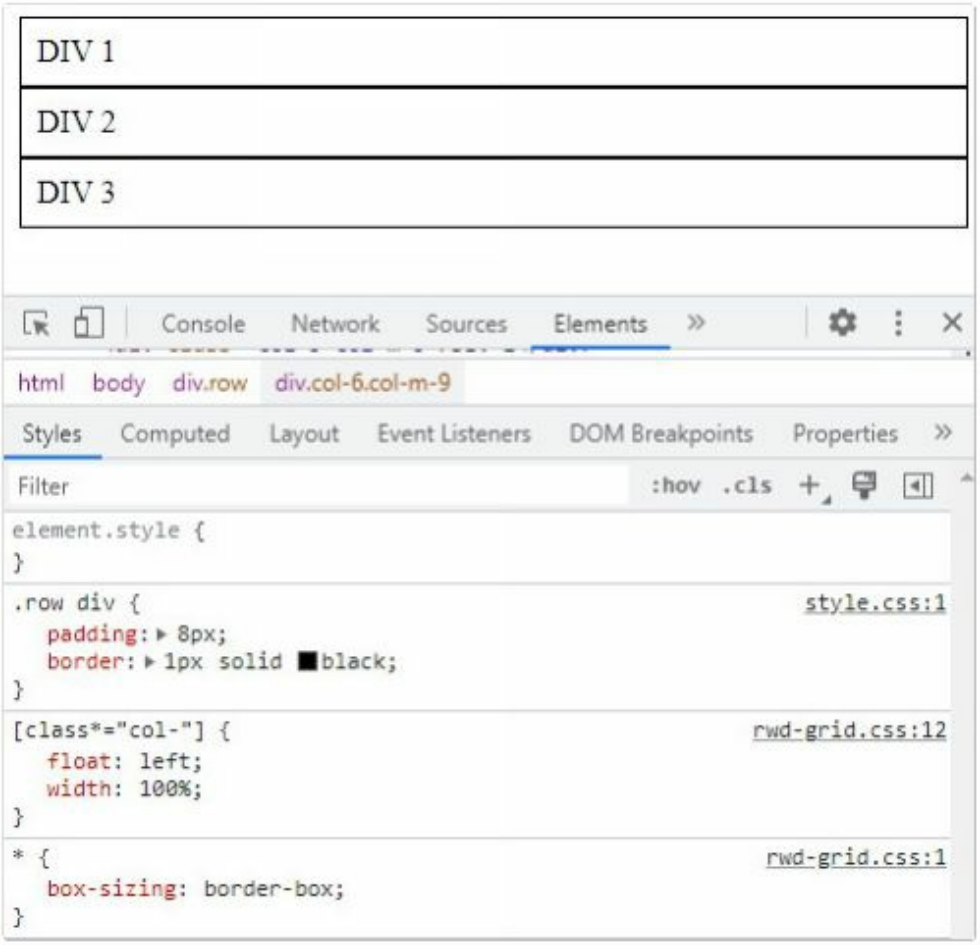
.5 Mobile-first vs. Desktop-first

.5.1 Mobile-First

Mobile-first responsive CSS is designed/written for smaller screen-size first. It will make the page display faster on smaller devices.

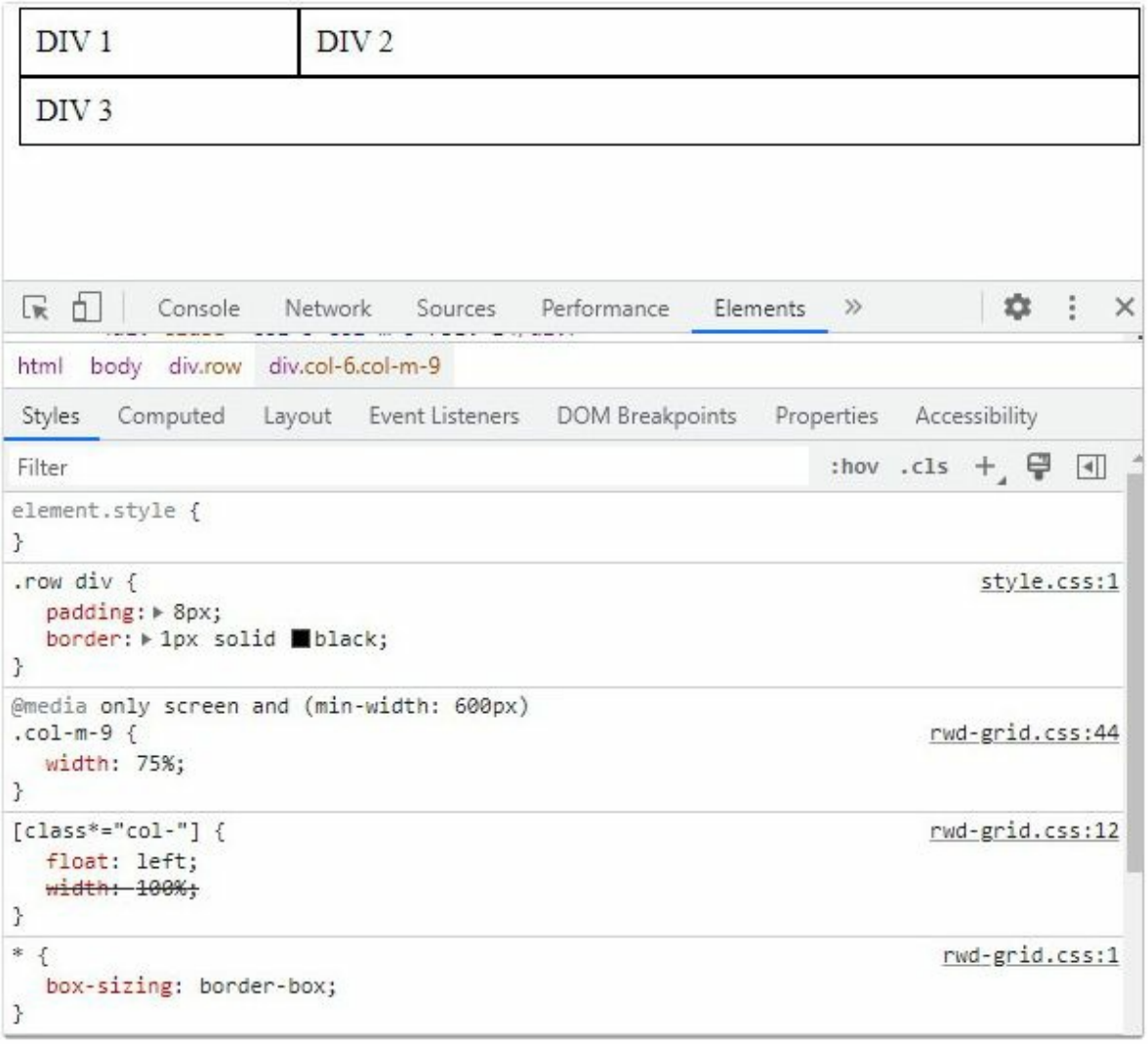
Mobile-first uses the “ min-width ” property in the media query so that the CSS inside it is only calculated when the screen size is bigger than the “ min-width .”

The example in the section “**RWD Grid View**” is mobile-first. Therefore, as in the screenshot, the browser only proceeds with a few CSS lines of code on a small device.



However, if we use a bigger device, more CSS must have proceeded, as

shown in the below screenshot.



2.5.2 Desktop-First

Desktop-first responsive CSS is designed/written for bigger screen size first. It will make the page display faster on bigger devices.

Desktop-first uses“ max-width ” property in the media query so that the CSS inside it is only calculated when the screen size is smaller than the “ max-width .”

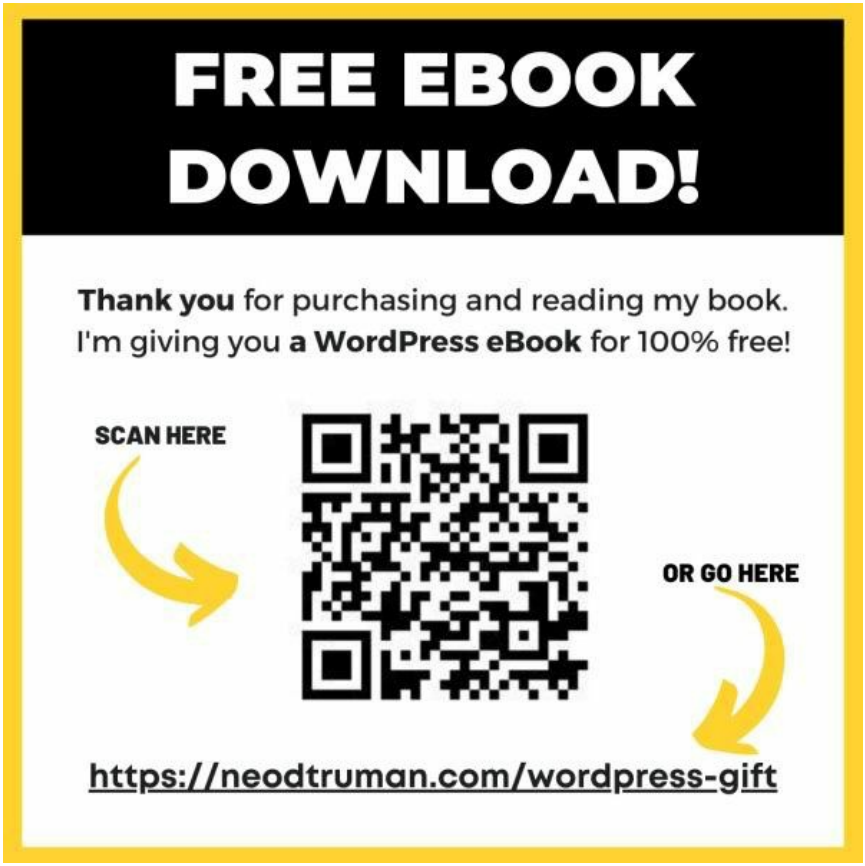
For example, let’s change the CSS code of the “RWD Grid View” to be desktop-first.

rwd-grid.css

```
* {
  box-sizing: border-box;
}

.row::after {
  content: "";
  clear: both;
  display: block;
}

[class*="col-"] {
  float: left;
}

/* For desktop: */
.col-1 { width: 8.33%; }
.col-2 { width: 16.66%; }
.col-3 { width: 25%; }
.col-4 { width: 33.33%; }
.col-5 { width: 41.66%; }
.col-6 { width: 50%; }
.col-7 { width: 58.33%; }
.col-8 { width: 66.66%; }
.col-9 { width: 75%; }
.col-10 { width: 83.33%; }
.col-11 { width: 91.66%; }
.col-12 { width: 100%; }

@media only screen and (max-width: 768px) {
```

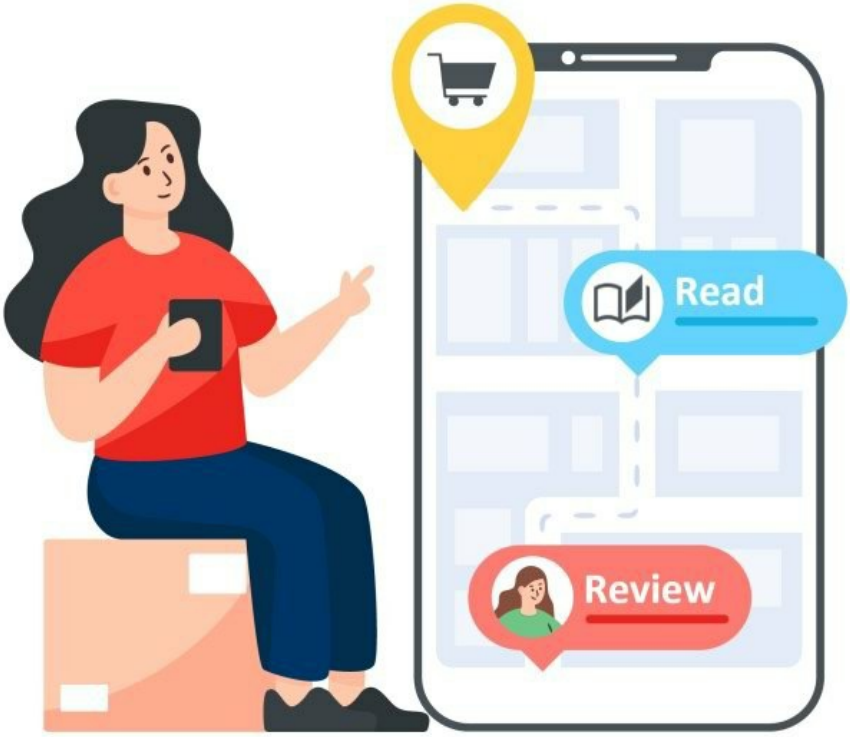
```
  /* For tablets: */
  .col-m-1 { width: 8.33%; }
  .col-m-2 { width: 16.66%; }
  .col-m-3 { width: 25%; }
  .col-m-4 { width: 33.33%; }
  .col-m-5 { width: 41.66%; }
  .col-m-6 { width: 50%; }
  .col-m-7 { width: 58.33%; }
  .col-m-8 { width: 66.66%; }
  .col-m-9 { width: 75%; }
  .col-m-10 { width: 83.33%; }
  .col-m-11 { width: 91.66%; }
  .col-m-12 { width: 100%; }
}
@media only screen and (max-width: 600px) {
  /* For mobile phones: */
  [class*="col-"] {
    width: 100%;
  }
}
```

BONUS: Free eBook Download



Please Leave a Review on Amazon

Thank you for purchasing and reading my book. Please leave a review on Amazon after reading this book.



This page has been left blank intentionally.
Please turn it over.

About the Author

Neo D. Truman has been using computers since 1998, when he was a child. In 2011, he got a Master of Science in Information Technology degree. **Neo** has more than 15 years of experience in software development, especially full-stack web development.

Other Books by The Author

HTML5 Coding Book for Beginners

https://www.amazon.com/dp/B0BGNL5XQY

CSS3 Coding Book for Beginners

https://www.amazon.com/dp/B0BN93R9F7

JavaScript Coding Book for Beginners

https://www.amazon.com/dp/B0BMSVSTJ7

React JS Application Development

https://www.amazon.com/dp/B0BGN8YFXK

Next JS, The Full Stack React JS Framework

https://www.amazon.com/dp/B0BH1YZW4R