** HTML is the standard markup language for Web pages.
** With HTML you can create your own Website.
** HTML is easy to learn - You will enjoy it!

Week 1-2: Introduction and Basics
Week 1:

1. HTML Introduction

HTML is the standard markup language for creating Web pages.

What is HTML?

- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

A Simple HTML Document

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Example Explained

- The `<!DOCTYPE html>` declaration defines that this document is an HTML5 document
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the HTML page
- The `<title>` element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The `<body>` element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag:

```
<tagname> Content goes here... </tagname>
```

The HTML **element** is everything from the start tag to the end tag:

```
<h1>My First Heading</h1>
<p>My first paragraph.</p>
```

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
<p>	My first paragraph.	</p>
<br>	none	none

Note: Some HTML elements have no content (like the <br> element). These elements are called empty elements. Empty elements do not have an end tag!

2. HTML Basic Examples

HTML Documents

All HTML documents must start with a document type declaration: `<!DOCTYPE html>`.

The HTML document itself begins with `<html>` and ends with `</html>`.

The visible part of the HTML document is between `<body>` and `</body>`.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

The <!DOCTYPE> Declaration

The `<!DOCTYPE>` declaration represents the document type, and helps browsers to display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The `<!DOCTYPE>` declaration is not case sensitive.

The `<!DOCTYPE>` declaration for HTML5 is:

```
<!DOCTYPE html>
```

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading:

Example

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the `<p>` tag:

Example

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

HTML Links

HTML links are defined with the `<a>` tag:

Example

```
<a href="https://www.facebook.com">This is a link</a>
```

The link's destination is specified in the `href` attribute.

Attributes are used to provide additional information about HTML elements.

You will learn more about attributes in a later chapter.

HTML Images

HTML images are defined with the `<img>` tag.

The source file (`src`), alternative text (`alt`), `width`, and `height` are provided as attributes:

Example

```
<img src="cat.jpg" alt="Cat" width="104" height="142">
```

How to View HTML Source

Have you ever seen a Web page and wondered "Hey! How did they do that?"

View HTML Source Code:

Click CTRL + U in an HTML page, or right-click on the page and select "View Page Source". This will open a new tab containing the HTML source code of the page.

Inspect an HTML Element:

Right-click on an element (or a blank area), and choose "Inspect" to see what elements are made up of (you will see both the HTML and the CSS). You can also edit the HTML or CSS on-the-fly in the Elements or Styles panel that opens.

Nested HTML Elements

HTML elements can be nested (this means that elements can contain other elements).

All HTML documents consist of nested HTML elements.

The following example contains four HTML elements (`<html>`, `<body>`, `<h1>` and `<p>`):

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Example Explained

The `<html>` element is the root element and it defines the whole HTML document.

It has a start tag `<html>` and an end tag `</html>`.

Then, inside the `<html>` element there is a `<body>` element:

```
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
```

The `<body>` element defines the document's body.

It has a start tag `<body>` and an end tag `</body>`.

Then, inside the `<body>` element there are two other elements: `<h1>` and `<p>`:

```
<h1>My First Heading</h1>
<p>My first paragraph.</p>
```

The `<h1>` element defines a heading.

It has a start tag `<h1>` and an end tag `</h1>`:

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

The `<p>` element defines a paragraph.

It has a start tag `<p>` and an end tag `</p>`:

```
<p>My first paragraph.</p>
```

Never Skip the End Tag

Some HTML elements will display correctly, even if you forget the end tag:

Example

```
<html>
<body>

<p>This is a paragraph
<p>This is a paragraph
```

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

Note: **However, never rely on this! Unexpected results and errors may occur if you forget the end tag!**

Empty HTML Elements

HTML elements with no content are called empty elements.

The `<br>` tag defines a line break, and is an empty element without a closing tag:

Example

```
<p>This is a <br> paragraph with a line break.</p>
```

HTML is Not Case Sensitive

HTML tags are not case sensitive: `<P>` means the same as `<p>`.

The HTML standard does not require lowercase tags but it is recommended for readability.

3. Attributes

HTML attributes provide additional information about HTML elements.

HTML Attributes

- All HTML elements can have **attributes**
- Attributes provide **additional information** about elements
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

The href Attribute

The `<a>` tag defines a hyperlink. The `href` attribute specifies the URL of the page the link goes to:

Example

```
<a href="https://www.facebook.com">Visit Facebook</a>
```

The src Attribute

The `<img>` tag is used to embed an image in an HTML page.
The `src` attribute specifies the path to the image to be displayed:

There are two ways to specify the URL in the `src` attribute:

1. Absolute URL - Links to an external image that is hosted on another website. Example: src="https://www.facebook.com/images/img_girl.jpg".

Notes: External images might be under copyright. If you do not get permission to use it, you may be in violation of copyright laws. In addition, you cannot control external images; it can suddenly be removed or changed.

2. Relative URL - Links to an image that is hosted within the website. Here, the URL does not include the domain name. If the URL begins without a slash, it will be relative to the current page. Example: src="img_girl.jpg". If the URL begins with a slash, it will be relative to the domain. Example: src="/images/img_girl.jpg".

Tip: It is almost always best to use relative URLs. They will not break if you change domain.

The width and height Attributes

The `<img>` tag should also contain the `width` and `height` attributes, which specify the width and height of the image (in pixels):

Example

```
<img src="img_girl.jpg" width="500" height="600">
```

The alt Attribute

The required `alt` attribute for the `<img>` tag specifies an alternate text for an image, if the image for some reason cannot be displayed. This can be due to a slow connection, or an error in the `src` attribute, or if the user uses a screen reader.

Example

```
<img src="img_girl.jpg" alt="Girl with a jacket">
```

Example

See what happens if we try to display an image that does not exist:

```
<img src="img_typo.jpg" alt="Girl with a jacket">
```

The style Attribute

The `style` attribute is used to add styles to an element, such as color, font, size, and more.

Example

```
<p style="color:red;">This is a red paragraph.</p>
```

The lang Attribute

You should always include the `lang` attribute inside the `<html>` tag, to declare the language of the Web page. This is meant to assist search engines and browsers.

The following example specifies English as the language:

```
<!DOCTYPE html>
<html lang="en">
<body>
...
</body>
</html>
```

Country codes can also be added to the language code in the `lang` attribute. So, the first two characters define the language of the HTML page, and the last two characters define the country.

The following example specifies English as the language and United States as the country:

```
<!DOCTYPE html>
<html lang="en-US">
<body>
...
</body>
</html>
```

The title Attribute

The `title` attribute defines some extra information about an element.

The value of the title attribute will be displayed as a tooltip when you mouse over the element:

Example

```
<p title="I'm a tooltip">This is a paragraph.</p>
```

NB: The title attribute (and all other attributes) can be written with uppercase or lowercase like **title** or **TITLE**.

Single or Double Quotes?

Double quotes around attribute values are the most common in HTML, but single quotes can also be used.

In some situations, when the attribute value itself contains double quotes, it is necessary to use single quotes:

```
<p title='John "ShotGun" Nelson'>
```

Or vice versa:

```
<p title="John 'ShotGun' Nelson">
```

Chapter Summary

- All HTML elements can have **attributes**
- The `href` attribute of `<a>` specifies the URL of the page the link goes to
- The `src` attribute of `<img>` specifies the path to the image to be displayed

- The `width` and `height` attributes of `<img>` provide size information for images
- The `alt` attribute of `<img>` provides an alternate text for an image
- The `style` attribute is used to add styles to an element, such as color, font, size, and more
- The `lang` attribute of the `<html>` tag declares the language of the Web page
- The `title` attribute defines some extra information about an element

Exercise:

Add a "tooltip" to the paragraph below with the text "About W3Schools".

```
<p  ="About Babcock">Babcock is a university.</p>
```

4. HTML Headings

HTML headings are titles or subtitles that you want to display on a webpage.

Example

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading.

Example

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

Note: Browsers automatically add some white space (a margin) before and after a heading.

Headings Are Important

Search engines use the headings to index the structure and content of your web pages.

Users often skim a page by its headings. It is important to use headings to show the document structure.

`<h1>` headings should be used for main headings, followed by `<h2>` headings, then the less important `<h3>`, and so on.

Note: Use HTML headings for headings only. Don't use headings to make text **BIG** or **bold**.

Bigger Headings

Each HTML heading has a default size. However, you can specify the size for any heading with the `style` attribute, using the CSS `font-size` property:

Example

```
<h1 style="font-size:60px;">Heading 1</h1>
```

Exercise:

Use the correct HTML tag to add a heading with the text "London".

```
<p>London is the capital city of England. It is the most
populous city in the United Kingdom, with a metropolitan
area of over 13 million inhabitants.</p>
```

5. HTML Paragraphs

A paragraph always starts on a new line, and is usually a block of text.

HTML Paragraphs

The HTML `<p>` element defines a paragraph.

A paragraph always starts on a new line, and browsers automatically add some white space (a margin) before and after a paragraph.

Example

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

HTML Display

You cannot be sure how HTML will be displayed.

Large or small screens, and resized windows will create different results.

With HTML, you cannot change the display by adding extra spaces or extra lines in your HTML code.

The browser will automatically remove any extra spaces and lines when the page is displayed:

Example

```
<p>
This paragraph
contains a lot of lines
in the source code,
but the browser
ignores it.
</p>

<p>
This paragraph
contains         a lot of spaces
in the source         code,
but the        browser
ignores it.
</p>
```

HTML Horizontal Rules

The `<hr>` tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

The `<hr>` element is used to separate content (or define a change) in an HTML page:

Example

```
<h1>This is heading 1</h1>
<p>This is some text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
<hr>
```

The `<hr>` tag is an empty tag, which means that it has no end tag.

HTML Line Breaks

The HTML `<br>` element defines a line break.

Use `<br>` if you want a line break (a new line) without starting a new paragraph:

Example

```
<p>This is<br>a paragraph<br>with line breaks.</p>
```

The `<br>` tag is an empty tag, which means that it has no end tag.

The Poem Problem

This poem will display on a single line:

Example

```
<p>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.

  Oh, bring back my Bonnie to me.
</p>
```

Solution - The HTML <pre> Element

The HTML `<pre>` element defines preformatted text.

The text inside a `<pre>` element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks:

Example

```
<pre>
  My Bonnie lies over the ocean.

  My Bonnie lies over the sea.

  My Bonnie lies over the ocean.
  Oh, bring back my Bonnie to me.
</pre>
```

Exercise:

Use the correct HTML tag to add a paragraph with the text "Hello World!".

```
<html>
<body>
    <!-- //-->
</body>
</html>
```

6. HTML Styles

The HTML `style` attribute is used to add styles to an element, such as color, font, size, and more.

Example

I am Red

I am Blue

I am Big

The HTML Style Attribute

Setting the style of an HTML element, can be done with the `style` attribute.

The HTML `style` attribute has the following syntax:

```
<tagname style="property:value;">
```

The **property** is a CSS property. The **value** is a CSS value.

You will learn more about CSS later in this tutorial.

Background Color

The CSS `background-color` property defines the background color for an HTML element.

Example

Set the background color for a page to powderblue:6

```
<body style="background-color:powderblue;">

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
```

Example

Set background color for two different elements:

```
<body>

<h1 style="background-color:powderblue;">This is a heading</h1>
<p style="background-color:tomato;">This is a paragraph.</p>

</body>
```

Text Color

The CSS `color` property defines the text color for an HTML element:

Example

```
<h1 style="color:blue;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
```

Fonts

The CSS `font-family` property defines the font to be used for an HTML element:

Example

```
<h1 style="font-family:verdana;">This is a heading</h1>
<p style="font-family:courier;">This is a paragraph.</p>
```

Text Size

The CSS font-size property defines the text size for an HTML element:

Example

```
<h1 style="font-size:300%;">This is a heading</h1>
<p style="font-size:160%;">This is a paragraph.</p>
```

Text Alignment

The CSS text-align property defines the horizontal text alignment for an HTML element:

Example

```
<h1 style="text-align:center;">Centered Heading</h1>
<p style="text-align:center;">Centered paragraph.</p>
```

Chapter Summary

- Use the style attribute for styling HTML elements
- Use background-color for background color
- Use color for text colors
- Use font-family for text fonts
- Use font-size for text sizes
- Use text-align for text alignment

Exercise:

Use the correct HTML attribute, and CSS, to set the color of the paragraph to "blue".

```
<p =";">This is a paragraph.</p>
```

7. HTML Text Formatting

HTML contains several elements for defining text with a special meaning.

Example

This text is bold

This text is italic

This is subscript and superscript

HTML Formatting Elements

Formatting elements were designed to display special types of text:

- `<b>` - Bold text
- `<strong>` - Important text
- `<i>` - Italic text
- `<em>` - Emphasized text
- `<mark>` - Marked text
- `<small>` - Smaller text
- `<del>` - Deleted text
- `<ins>` - Inserted text
- `<sub>` - Subscript text
- `<sup>` - Superscript text

HTML <b> and <strong> Elements

The HTML `<b>` element defines bold text, without any extra importance.

Example

```
<b>This text is bold</b>
```

The HTML `<strong>` element defines text with strong importance. The content inside is typically displayed in bold.

Example

```
<strong>This text is important!</strong>
```

HTML <i> and <em> Elements

The HTML `<i>` element defines a part of text in an alternate voice or mood. The content inside is typically displayed in italic.

Tip: The `<i>` tag is often used to indicate a technical term, a phrase from another language, a thought, a ship name, etc.

Example

```
<i>This text is italic</i>
```

The HTML `<em>` element defines emphasized text. The content inside is typically displayed in italic.

Tip: A screen reader will pronounce the words in `<em>` with an emphasis, using verbal stress.

Example

```
<em>This text is emphasized</em>
```

HTML <small> Element

The HTML `<small>` element defines smaller text:

Example

```
<small>This is some smaller text.</small>
```

HTML <mark> Element

The HTML `<mark>` element defines text that should be marked or highlighted:

Example

```
<p>Do not forget to buy <mark>milk</mark> today.</p>
```

HTML <del> Element

The HTML `<del>` element defines text that has been deleted from a document. Browsers will usually strike a line through deleted text:

Example

```
<p>My favorite color is <del>blue</del> red.</p>
```

HTML <ins> Element

The HTML `<ins>` element defines a text that has been inserted into a document. Browsers will usually underline inserted text:

Example

```
<p>My favorite color is <del>blue</del> <ins>red</ins>.</p>
```

HTML <sub> Element

The HTML `<sub>` element defines subscript text. Subscript text appears half a character below the normal line, and is sometimes rendered in a smaller font. Subscript text can be used for chemical formulas, like H_2O:

Example

```
<p>This is <sub>subscripted</sub> text.</p>
```

HTML <sup> Element

The HTML `<sup>` element defines superscript text. Superscript text appears half a character above the normal line, and is sometimes rendered in a smaller font. Superscript text can be used for footnotes, like WWW[1]:

Example

```
<p>This is <sup>superscripted</sup> text.</p>
```

Exercise:

Add extra importance to the word "degradation" in the paragraph below.

```
<p>
WWF's mission is to stop the degradation of our planet's
natural environment.
</p>
```

HTML Text Formatting Elements

Tag	Description
<b>	Defines bold text
<em>	Defines emphasized text
<i>	Defines a part of text in an alternate voice or mood
<small>	Defines smaller text
<strong>	Defines important text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text
<mark>	Defines marked/highlighted text

Week 2:

8. HTML Comments

HTML comments are not displayed in the browser, but they can help document your HTML source code.

HTML Comment Tag

You can add comments to your HTML source by using the following syntax:

```
<!-- Write your comments here -->
```

Notice that there is an exclamation point (!) in the start tag, but not in the end tag.

Note: Comments are not displayed by the browser, but they can help document your HTML source code.

Add Comments

With comments you can place notifications and reminders in your HTML code:

Example

```
<!-- This is a comment -->

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

<!-- Remember to add more information here -->
```

Hide Content

Comments can be used to hide content.

This can be helpful if you hide content temporarily:

Example

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

<!-- <p>This is another paragraph </p> -->

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

You can also hide more than one line. Everything between the `<!--` and the `-->` will be hidden from the display.

Example

Hide a section of HTML code:

```
<p>This is a paragraph.</p>
<!--
<p>Look at this cool image:</p>
<img border="0" src="pic_trulli.jpg" alt="Trulli">
-->
<p>This is a paragraph too.</p>
```

Comments are also great for debugging HTML, because you can comment out HTML lines of code, one at a time, to search for errors.

Hide Inline Content

Comments can be used to hide parts in the middle of the HTML code.

Example

Hide a part of a paragraph:

```
<p>This <!-- great text --> is a paragraph.</p>
```

Exercise:

Use the HTML comment tag to make a comment out of the "This is a comment" text.

```
<h1>This is a heading</h1>
 This is a comment
<p>This is a paragraph.</p>
```

9. HTML Colors

HTML colors are specified with predefined color names, or with RGB, HEX, HSL, RGBA, or HSLA values.

Color Names

In HTML, a color can be specified by using a color name:

Tomato
Orange
DodgerBlue
MediumSeaGreen
Gray
SlateBlue
Violet
LightGray

Background Color

You can set the background color for HTML elements:

Hello World

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.

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

Text Color

You can set the color of text:

Hello World

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.

Example

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

Border Color

You can set the color of borders:

Hello World

Hello World

Hello World

Example

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>
```

Color Values

In HTML, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values.

The following three <div> elements have their background color set with RGB, HEX, and HSL values:

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

The following two <div> elements have their background color set with RGBA and HSLA values, which add an Alpha channel to the color (here we have 50% transparency):

Example

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>
```

9(i). HTML RGB and RGBA Colors

An RGB color value represents RED, GREEN, and BLUE light sources.

An RGBA color value is an extension of RGB with an Alpha channel (opacity).

RGB Color Values

In HTML, a color can be specified as an RGB value, using this formula:

`rgb(red, green, blue)`

Each parameter (red, green, and blue) defines the intensity of the color with a value between 0 and 255.

This means that there are 256 x 256 x 256 = 16777216 possible colors!

For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255), and the other two (green and blue) are set to 0.

Another example, rgb(0, 255, 0) is displayed as green, because green is set to its highest value (255), and the other two (red and blue) are set to 0.

To display black, set all color parameters to 0, like this: rgb(0, 0, 0).

To display white, set all color parameters to 255, like this: rgb(255, 255, 255).

Experiment by mixing the RGB values below:`rgb(255, 99, 71)`

```
<h1 style="background-color:rgb(255, 0, 0);">rgb(255, 0, 0)</h1>
<h1 style="background-color:rgb(0, 0, 255);">rgb(0, 0, 255)</h1>
<h1 style="background-color:rgb(60, 179, 113);">rgb(60, 179,
113)</h1>
<h1 style="background-color:rgb(238, 130, 238);">rgb(238, 130,
238)</h1>
<h1 style="background-color:rgb(255, 165, 0);">rgb(255, 165, 0)</h1>
<h1 style="background-color:rgb(106, 90, 205);">rgb(106, 90,
205)</h1>
```

Shades of Gray

Shades of gray are often defined using equal values for all three parameters:

<h1 style="background-color:rgb(60, 60, 60);">rgb(60, 60, 60)</h1>

<h1 style="background-color:rgb(100, 100, 100);">rgb(100, 100, 100)</h1>

<h1 style="background-color:rgb(140, 140, 140);">rgb(140, 140, 140)</h1>

<h1 style="background-color:rgb(180, 180, 180);">rgb(180, 180, 180)</h1>

<h1 style="background-color:rgb(200, 200, 200);">rgb(200, 200, 200)</h1>

<h1 style="background-color:rgb(240, 240, 240);">rgb(240, 240, 240)</h1>

RGBA Color Values

RGBA color values are an extension of RGB color values with an Alpha channel - which specifies the opacity for a color.

An RGBA color value is specified with:

`rgba(red, green, blue, alpha)`

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

Experiment by mixing the RGBA values below:

<h1 style="background-color:rgba(255, 99, 71, 0);">rgba(255, 99, 71, 0)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.2);">rgba(255, 99, 71, 0.2)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.4);">rgba(255, 99, 71, 0.4)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.6);">rgba(255, 99, 71, 0.6)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.8);">rgba(255, 99, 71, 0.8)</h1>

<h1 style="background-color:rgba(255, 99, 71, 1);">rgba(255, 99, 71, 1)</h1>

9(ii). HTML HEX Colors

A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color.

HEX Color Values

In HTML, a color can be specified using a hexadecimal value in the form:

#rrggbb

Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

For example, #ff0000 is displayed as red, because red is set to its highest value (ff), and the other two (green and blue) are set to 00.

Another example, #00ff00 is displayed as green, because green is set to its highest value (ff), and the other two (red and blue) are set to 00.

To display black, set all color parameters to 00, like this: #000000.

To display white, set all color parameters to ff, like this: #ffffff.

Experiment by mixing the HEX values below:

```
<h1 style="background-color:#ff0000;">#ff0000</h1>
<h1 style="background-color:#0000ff;">#0000ff</h1>
<h1 style="background-color:#3cb371;">#3cb371</h1>
<h1 style="background-color:#ee82ee;">#ee82ee</h1>
<h1 style="background-color:#ffa500;">#ffa500</h1>
<h1 style="background-color:#6a5acd;">#6a5acd</h1>
```

Shades of Gray

Shades of gray are often defined using equal values for all three parameters:

```
<h1 style="background-color:#404040;">#404040</h1>
<h1 style="background-color:#686868;">#686868</h1>
<h1 style="background-color:#a0a0a0;">#a0a0a0</h1>
<h1 style="background-color:#bebebe;">#bebebe</h1>
<h1 style="background-color:#dcdcdc;">#dcdcdc</h1>
<h1 style="background-color:#f8f8f8;">#f8f8f8</h1>
```

9(iii). HTML HSL and HSLA Colors

HSL stands for hue, saturation, and lightness.

HSLA color values are an extension of HSL with an Alpha channel (opacity).

HSL Color Values

In HTML, a color can be specified using hue, saturation, and lightness (HSL) in the form:

`hsl(hue, saturation, lightness)`

Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.

Saturation is a percentage value. 0% means a shade of gray, and 100% is the full color.

Lightness is also a percentage value. 0% is black, and 100% is white.

Experiment by mixing the HSL values below:

```
<h1 style="background-color:hsl(0, 100%, 50%);">hsl(0, 100%,
50%)</h1>
<h1 style="background-color:hsl(240, 100%, 50%);">hsl(240, 100%,
50%)</h1>
<h1 style="background-color:hsl(147, 50%, 47%);">hsl(147, 50%,
47%)</h1>
<h1 style="background-color:hsl(300, 76%, 72%);">hsl(300, 76%,
72%)</h1>
<h1 style="background-color:hsl(39, 100%, 50%);">hsl(39, 100%,
50%)</h1>
```

<h1 style="background-color:hsl(248, 53%, 58%);">hsl(248, 53%, 58%)</h1>

Saturation

Saturation can be described as the intensity of a color.

100% is pure color, no shades of gray.

50% is 50% gray, but you can still see the color.

0% is completely gray; you can no longer see the color.

```
<h1 style="background-color:hsl(0, 100%, 50%);">hsl(0,
100%, 50%)</h1>

<h1 style="background-color:hsl(0, 80%, 50%);">hsl(0, 80%,
50%)</h1>

<h1 style="background-color:hsl(0, 60%, 50%);">hsl(0, 60%,
50%)</h1>

<h1 style="background-color:hsl(0, 40%, 50%);">hsl(0, 40%,
50%)</h1>

<h1 style="background-color:hsl(0, 20%, 50%);">hsl(0, 20%,
50%)</h1>

<h1 style="background-color:hsl(0, 0%, 50%);">hsl(0, 0%,
50%)</h1>



<p>With HSL colors, less saturation mean less color. 0% is
completely gray.</p>
```

Lightness

The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light), and 100% means full lightness (white).

Example

```
<h1 style="background-color:hsl(0, 100%, 0%);">hsl(0, 100%, 0%)</h1>
<h1 style="background-color:hsl(0, 100%, 25%);">hsl(0, 100%,
25%)</h1>
<h1 style="background-color:hsl(0, 100%, 50%);">hsl(0, 100%,
50%)</h1>
<h1 style="background-color:hsl(0, 100%, 75%);">hsl(0, 100%,
75%)</h1>
<h1 style="background-color:hsl(0, 100%, 90%);">hsl(0, 100%,
90%)</h1>
<h1 style="background-color:hsl(0, 100%, 100%);">hsl(0, 100%,
100%)</h1>

<p>With HSL colors, 0% lightness means black, and 100 lightness
means white.</p>
```

Shades of Gray

Shades of gray are often defined by setting the hue and saturation to 0, and adjusting the lightness from 0% to 100% to get darker/lighter shades:

Example

```
<h1 style="background-color:hsl(0, 0%, 20%);">hsl(0, 0%, 20%)</h1>
<h1 style="background-color:hsl(0, 0%, 30%);">hsl(0, 0%, 30%)</h1>
<h1 style="background-color:hsl(0, 0%, 40%);">hsl(0, 0%, 40%)</h1>
<h1 style="background-color:hsl(0, 0%, 60%);">hsl(0, 0%, 60%)</h1>
<h1 style="background-color:hsl(0, 0%, 70%);">hsl(0, 0%, 70%)</h1>
<h1 style="background-color:hsl(0, 0%, 90%);">hsl(0, 0%, 90%)</h1>
```

HSLA Color Values

HSLA color values are an extension of HSL color values, with an Alpha channel - which specifies the opacity for a color.

An HSLA color value is specified with:

`hsla(hue, saturation, lightness, alpha)`

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

Experiment by mixing the HSLA values below:

<h1 style="background-color:hsla(9, 100%, 64%, 0);">hsla(9, 100%, 64%, 0)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.2);">hsla(9, 100%, 64%, 0.2)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.4);">hsla(9, 100%, 64%, 0.4)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.6);">hsla(9, 100%, 64%, 0.6)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.8);">hsla(9, 100%, 64%, 0.8)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 1);">hsla(9, 100%, 64%, 1)</h1>

10. HTML Links

Links are found in nearly all web pages. Links allow users to click their way from page to page.

HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. A link can be an image or any other HTML element!

HTML Links - Syntax

The HTML `<a>` tag defines a hyperlink. It has the following syntax:

```
<a href="url">link text</a>
```

The most important attribute of the `<a>` element is the `href` attribute, which indicates the link's destination.

The link text is the part that will be visible to the reader.

Clicking on the link text, will send the reader to the specified URL address.

Example

This example shows how to create a link to W3Schools.com:

```
<a href="https://www.facebook.com/">Visit W3Schools.com!</a>
```

By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

Tip: Links can of course be styled with CSS, to get another look!

HTML Links - The target Attribute

By default, the linked page will be displayed in the current browser window. To change this, you must specify another target for the link.

The `target` attribute specifies where to open the linked document.

The `target` attribute can have one of the following values:

- `_self` - Default. Opens the document in the same window/tab as it was clicked
- `_blank` - Opens the document in a new window or tab
- `_parent` - Opens the document in the parent frame
- `_top` - Opens the document in the full body of the window

Example

Use target="_blank" to open the linked document in a new browser window or tab:

```
<a href="https://www.facebook.com/" target="_blank">Visit Facebook!</a>
```

By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

Tip: Links can of course be styled with CSS, to get another look!

HTML Links - The target Attribute

By default, the linked page will be displayed in the current browser window. To change this, you must specify another target for the link.

The `target` attribute specifies where to open the linked document.

The `target` attribute can have one of the following values:

- `_self` - Default. Opens the document in the same window/tab as it was clicked
- `_blank` - Opens the document in a new window or tab
- `_parent` - Opens the document in the parent frame

- _top - Opens the document in the full body of the window

Example

Use target="_blank" to open the linked document in a new browser window or tab:

```
<a href="https://www.w3schools.com/" target="_blank">Visit
W3Schools!</a>
```

Absolute URLs vs. Relative URLs

Both examples above are using an **absolute URL** (a full web address) in the href attribute.

A local link (a link to a page within the same website) is specified with a **relative URL** (without the "https://www" part):

Example

```
<h2>Absolute URLs</h2>
<p><a href="https://www.w3.org/">W3C</a></p>
<p><a href="https://www.google.com/">Google</a></p>

<h2>Relative URLs</h2>
<p><a href="html_images.asp">HTML Images</a></p>
<p><a href="/css/default.asp">CSS Tutorial</a></p>
```

HTML Links - Use an Image as a Link

To use an image as a link, just put the <img> tag inside the <a> tag:

Example

```
<a href="default.asp">
<img src="smiley.gif" alt="HTML
```

```
tutorial" style="width:42px;height:42px;">
</a>
```

Link to an Email Address

Use `mailto:` inside the `href` attribute to create a link that opens the user's email program (to let them send a new email):

Example

```
<a href="mailto:someone@example.com">Send email</a>
```

Button as a Link

To use an HTML button as a link, you have to add some JavaScript code.

JavaScript allows you to specify what happens at certain events, such as a click of a button:

Example

```
<button onclick="document.location='default.asp'">HTML
Tutorial</button>
```

Link Titles

The `title` attribute specifies extra information about an element. The information is most often shown as a tooltip text when the mouse moves over the element.

Example

```
<a href="https://www.w3schools.com/html/" title="Go to W3Schools HTML
section">Visit our HTML Tutorial</a>
```

More on Absolute URLs and Relative URLs

Example

Use a full URL to link to a web page:

```
<a href="https://www.w3schools.com/html/default.asp">HTML tutorial</a>
```

Example

Link to a page located in the html folder on the current web site:

```
<a href="/html/default.asp">HTML tutorial</a>
```

Example

Link to a page located in the same folder as the current page:

Chapter Summary

- Use the `<a>` element to define a link
- Use the `href` attribute to define the link address
- Use the `target` attribute to define where to open the linked document
- Use the `<img>` element (inside `<a>`) to use an image as a link
- Use the `mailto:` scheme inside the `href` attribute to create a link that opens the user's email program

HTML Link Tags

Tag	Description
<a>	Defines a hyperlink

10(i). HTML Links - Different Colors

An HTML link is displayed in a different color depending on whether it has been visited, is unvisited, or is active.

HTML Link Colors

By default, a link will appear like this (in all browsers):

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

You can change the link state colors, by using CSS:

Example

Here, an unvisited link will be green with no underline. A visited link will be pink with no underline. An active link will be yellow and underlined. In addition, when mousing over a link (a:hover) it will become red and underlined:

```
<style>
a:link {
  color: green;
  background-color: transparent;
  text-decoration: none;
}

a:visited {
  color: pink;
  background-color: transparent;
  text-decoration: none;
}

a:hover {
  color: red;
  background-color: transparent;
  text-decoration: underline;
}

a:active {
  color: yellow;
  background-color: transparent;
  text-decoration: underline;
}
</style>
```

Link Buttons

A link can also be styled as a button, by using CSS:

This is a link

Example

```
<style>
a:link, a:visited {
  background-color: #f44336;
  color: white;
  padding: 15px 25px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
}

a:hover, a:active {
  background-color: red;
}
</style>
```

10(ii). HTML Links - Create Bookmarks

HTML links can be used to create bookmarks, so that readers can jump to specific parts of a web page.

Create a Bookmark in HTML

Bookmarks can be useful if a web page is very long.

To create a bookmark - first create the bookmark, then add a link to it.

When the link is clicked, the page will scroll down or up to the location with the bookmark.

Example

First, use the `id` attribute to create a bookmark:

```
<h2 id="C4">Chapter 4</h2>
```

Then, add a link to the bookmark ("Jump to Chapter 4"), from within the same page:

Example

```
<a href="#C4">Jump to Chapter 4</a>
```

You can also add a link to a bookmark on another page:

```
<a href="html_demo.html#C4">Jump to Chapter 4</a>
```

Chapter Summary

- Use the `id` attribute (id="value") to define bookmarks in a page
- Use the `href` attribute (href="#value") to link to the bookmark

Exercise:

Use the correct HTML to make the text below into a link to "default.html".

11. HTML Images

Images can improve the design and the appearance of a web page.

Example

```
<img src="pic_trulli.jpg" alt="Italian Trulli">
```

Example

```
<img src="img_girl.jpg" alt="Girl in a jacket">
```

Example

```
<img src="img_chania.jpg" alt="Flowers in Chania">
```

HTML Images Syntax

The HTML `<img>` tag is used to embed an image in a web page.

Images are not technically inserted into a web page; images are linked to web pages. The `<img>` tag creates a holding space for the referenced image.

The `<img>` tag is empty, it contains attributes only, and does not have a closing tag.

The `<img>` tag has two required attributes:

- src - Specifies the path to the image
- alt - Specifies an alternate text for the image

Syntax

```
<img src="url" alt="alternatetext">
```

The src Attribute

The required `src` attribute specifies the path (URL) to the image.

Note: When a web page loads, it is the browser, at that moment, that gets the image from a web server and inserts it into the page. Therefore, make sure that the image actually stays in the same spot in relation to the web page, otherwise your visitors will get a broken link icon. The broken link icon and the alt text are shown if the browser cannot find the image.

Example

```
<img src="img_chania.jpg" alt="Flowers in Chania">
```

The alt Attribute

The required alt attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the alt attribute should describe the image:

Example

```
<img src="img_chania.jpg" alt="Flowers in Chania">
```

If a browser cannot find an image, it will display the value of the alt attribute:

Example

```
<img src="wrongname.gif" alt="Flowers in Chania">
```

Tip: A screen reader is a software program that reads the HTML code, and allows the user to "listen" to the content. Screen readers are useful for people who are visually impaired or learning disabled.

Image Size - Width and Height

You can use the style attribute to specify the width and height of an image.

Example

```
<img src="img_girl.jpg" alt="Girl in a
jacket" style="width:500px;height:600px;">
```

Alternatively, you can use the `width` and `height` attributes:

Example

```
<img src="img_girl.jpg" alt="Girl in a
jacket" width="500" height="600">
```

The `width` and `height` attributes always define the width and height of the image in pixels.

Note: Always specify the width and height of an image. If width and height are not specified, the web page might flicker while the image loads.

Width and Height, or Style?

The `width`, `height`, and `style` attributes are all valid in HTML.

However, we suggest using the `style` attribute. It prevents styles sheets from changing the size of images:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
  width: 100%;
}
</style>
</head>
<body>

<img src="html5.gif" alt="HTML5 Icon" width="128" height="128">
```

```
<img src="html5.gif" alt="HTML5
Icon" style="width:128px;height:128px;">

</body>
</html>
```

Images in Another Folder

If you have your images in a sub-folder, you must include the folder name in the `src` attribute:

Example

```
<img src="/images/html5.gif" alt="HTML5
Icon" style="width:128px;height:128px;">
```

Images on Another Server/Website

Some web sites point to an image on another server.

To point to an image on another server, you must specify an absolute (full) URL in the `src` attribute:

Example

```
<img src="https://www.facebook.com/images/facebook_green.jpg" alt="
Facebook.com">
```

Notes on external images: External images might be under copyright. If you do not get permission to use it, you may be in violation of copyright laws. In addition, you cannot control external images; they can suddenly be removed or changed.

Animated Images

HTML allows animated GIFs:

Example

```
<img src="programming.gif" alt="Computer
Man" style="width:48px;height:48px;">
```

Image as a Link

To use an image as a link, put the `<img>` tag inside the `<a>` tag:

Example

```
<a href="default.asp">
  <img src="smiley.gif" alt="HTML
tutorial" style="width:42px;height:42px;">
</a>
```

Image Floating

Use the CSS `float` property to let the image float to the right or to the left of a text:

Example

```
<p><img src="smiley.gif" alt="Smiley
face" style="float:right;width:42px;height:42px;">
The image will float to the right of the text.</p>

<p><img src="smiley.gif" alt="Smiley
face" style="float:left;width:42px;height:42px;">
The image will float to the left of the text.</p>
```

Common Image Formats

Here are the most common image file types, which are supported in all browsers (Chrome, Edge, Firefox, Safari, Opera):

Abbreviation	File Format	File Extension

APNG	Animated Portable Network Graphics	.apng
GIF	Graphics Interchange Format	.gif
ICO	Microsoft Icon	.ico, .cur
JPEG	Joint Photographic Expert Group image	.jpg, .jpeg, .jfif, .pjpeg, .pjp
PNG	Portable Network Graphics	.png
SVG	Scalable Vector Graphics	.svg

Chapter Summary

- Use the HTML `<img>` element to define an image
- Use the HTML `src` attribute to define the URL of the image
- Use the HTML `alt` attribute to define an alternate text for an image, if it cannot be displayed
- Use the HTML `width` and `height` attributes or the CSS `width` and `height` properties to define the size of the image
- Use the CSS `float` property to let the image float to the left or to the right

Note: Loading large images takes time, and can slow down your web page. Use images carefully.

HTML Exercises

Exercise:

Use the HTML image attributes to set the size of the image to 250 pixels wide and 400 pixels tall.

```
<img src="scream.png"    ="250"    ="400">
```

HTML Image Tags

Tag	Description
<img>	Defines an image
<map>	Defines an image map
<area>	Defines a clickable area inside an image map
<picture>	Defines a container for multiple image resources

12. HTML Background Images

A background image can be specified for almost any HTML element.

Background Image on a HTML element

To add a background image on an HTML element, use the HTML `style` attribute and the CSS `background-image` property:

Example

Add a background image on a HTML element:

```
<p style="background-image: url('img_girl.jpg');">
```

You can also specify the background image in the `<style>` element, in the `<head>` section:

Example

Specify the background image in the `<style>` element:

```
<style>
p {
  background-image: url('img_girl.jpg');
}
</style>
```

Background Image on a Page

If you want the entire page to have a background image, you must specify the background image on the `<body>` element:

Example

Add a background image for the entire page:

```
<style>
body {
  background-image: url('img_girl.jpg');
}
</style>
```

Background Repeat

If the background image is smaller than the element, the image will repeat itself, horizontally and vertically, until it reaches the end of the element:

Example

```
<style>
body {
  background-image: url('example_img_girl.jpg');
}
</style>
```

To avoid the background image from repeating itself, set the `background-repeat` property to `no-repeat`.

Example

```
<style>
body {
  background-image: url('example_img_girl.jpg');
  background-repeat: no-repeat;
}
</style>
```

Background Cover

If you want the background image to cover the entire element, you can set the `background-size` property to `cover.`

Also, to make sure the entire element is always covered, set the `background-attachment` property to `fixed:`

This way, the background image will cover the entire element, with no stretching (the image will keep its original proportions):

Example

```
<style>
body {
  background-image: url('img_girl.jpg');
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: cover;
}
</style>
```

Background Stretch

If you want the background image to stretch to fit the entire element, you can set the `background-size` property to `100% 100%`:

Try resizing the browser window, and you will see that the image will stretch, but always cover the entire element.

Example

```
<style>
body {
  background-image: url('img_girl.jpg');
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: 100% 100%;
}
</style>
```

12(i). HTML <picture> Element

The HTML `<picture>` element allows you to display different pictures for different devices or screen sizes.

The HTML <picture> Element

The HTML `<picture>` element gives web developers more flexibility in specifying image resources.

The `<picture>` element contains one or more `<source>` elements, each referring to different images through the `srcset` attribute. This way the browser can choose the image that best fits the current view and/or device.

Each `<source>` element has a `media` attribute that defines when the image is the most suitable.

Example

Show different images for different screen sizes:

```
<picture>
  <source media="(min-width: 650px)" srcset="img_food.jpg">
  <source media="(min-width: 465px)" srcset="img_car.jpg">
  <img src="img_girl.jpg">
</picture>
```

Note: Always specify an `<img>` element as the last child element of the `<picture>` element. The `<img>` element is used by browsers that do not support the `<picture>` element, or if none of the `<source>` tags match.

When to use the Picture Element

There are two main purposes for the `<picture>` element:

1. Bandwidth

If you have a small screen or device, it is not necessary to load a large image file. The browser will use the first `<source>` element with matching attribute values, and ignore any of the following elements.

2. Format Support

Some browsers or devices may not support all image formats. By using the `<picture>` element, you can add images of all formats, and the browser will use the first format it recognizes, and ignore any of the following elements.

Example

The browser will use the first image format it recognizes:

```
<picture>
  <source srcset="img_avatar.png">
  <source srcset="img_girl.jpg">
  <img src="img_beatles.gif" alt="Beatles" style="width:auto;">
</picture>
```

Note: The browser will use the first `<source>` element with matching attribute values, and ignore any following `<source>` elements.

HTML Image Tags

Tag	Description
<img>	Defines an image
<map>	Defines an image map
<area>	Defines a clickable area inside an image map
<picture>	Defines a container for multiple image resources

13. CSS Introduction

CSS is the language we use to style a Web page.

What is CSS?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

Why Use CSS?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

CSS Example

```css
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

CSS Solved a Big Problem

HTML was NEVER intended to contain tags for formatting a web page!

HTML was created to describe the content of a web page, like:

<h1>This is a heading</h1>

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

When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.

To solve this problem, the World Wide Web Consortium (W3C) created CSS.

CSS removed the style formatting from the HTML page!

CSS Saves a Lot of Work!

The style definitions are normally saved in external .css files.

With an external stylesheet file, you can change the look of an entire website by changing just one file!

14. CSS Selectors

A CSS selector selects the HTML element(s) you want to style.

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

This page will explain the most basic CSS selectors.

The CSS element Selector

The element selector selects HTML elements based on the element name.

Example

Here, all <p> elements on the page will be center-aligned, with a red text color:

```
p {
  text-align: center;
  color: red;
}
```

The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

The CSS rule below will be applied to the HTML element with id="para1":

```css
#para1 {
  text-align: center;
  color: red;
}
```

Note: An id name cannot start with a number!

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

Example

In this example all HTML elements with class="center" will be red and center-aligned:

```css
.center {
  text-align: center;
  color: red;
}
```

You can also specify that only specific HTML elements should be affected by a class.

Example

In this example only <p> elements with class="center" will be red and center-aligned:

```
p.center {
  text-align: center;
  color: red;
}
```

HTML elements can also refer to more than one class.

Example

In this example the <p> element will be styled according to class="center" and to class="large":

```
<p class="center large">This paragraph refers to two classes.</p>
```

Note: A class name cannot start with a number!

The CSS Universal Selector

The universal selector (*) selects all HTML elements on the page.

Example

The CSS rule below will affect every HTML element on the page:

```
* {
  text-align: center;
  color: blue;
}
```

The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

```
h1 {
  text-align: center;
  color: red;
}

h2 {
  text-align: center;
  color: red;
}

p {
  text-align: center;
  color: red;
}
```

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

Example

In this example we have grouped the selectors from the code above:

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

All CSS Simple Selectors

Selector	Example	Example description

#id	#firstname	Selects the element with id="firstname"
.class	.intro	Selects all elements with class="intro"
element.class	p.intro	Selects only <p> elements with class="intro"
*	*	Selects all elements
element	p	Selects all <p> elements
element,element,..	div, p	Selects all <div> elements and all <p> elements

15. CSS PROPERTIES

A CSS property assign a style or behavior to an HTML element.

Examples include: `color`, `border`, `margin`, `font-style`, and `transform`.

Example

A table header with a custom `background-color`.

```
<style>

  table.tb { width: 450px; border-collapse: collapse; }

  .tb th, .tb td { border: solid 1px #777; padding: 5px; }

  .tb thead { background-color: steelblue; color: white; }

</style>

<table class="tb">
```

```
  <thead>

    <tr>

      <th>First Name</th>

      <th>Last Name</th>

      <th>Country</th>



    </tr>

  </thead>

  <tbody>

    <tr>

      <td>Denice</td>

      <td>Hobermann</td>

      <td>Canada</td>

    </tr>

    <tr>

      <td>Paulo</td>

      <td>Cornell</td>

      <td>Brazil</td>

    </tr>
    <tr>

      <td>Jane</td>

      <td>Hollander</td>

      <td>USA</td>

    </tr>

  </tbody>

</table>
```

Property List

CSS supports more than 200 CSS properties. Here's a complete list.
Click a name for details.

PROPERTY	DESCRIPTION
`align-content`	Aligns items in a flex container along flex lines.
`align-items`	Aligns evenly spaced items in a flex container.
`align-self`	Aligns an item inside a flex container.
`all`	Resets all element properties to its default or inherited values.
`animation`	Creates an animating element.
`animation-delay`	Sets a delay before an animation begins.
`animation-direction`	Sets how, in which direction, an animation is played.
`animation-duration`	Defines the duration of an animation cycle.
`animation-fill-mode`	Defines how styles are applied before and

	after animation.
`animation-iteration-count`	Sets the number of times an animation is played.
`animation-name`	Defines a name for the animation.
`animation-play-state`	Sets the animation play state to running or paused.
`animation-timing-function`	Specifies the animation speed curve.
`backface-visibility`	Shows or hides the backface visibility of an element.
`background`	Sets the background of an element.
`background-attachment`	Defines how the background is attached to an element.
`background-blend-mode`	Defines the background layer blending mode.
`background-clip`	Defines how background extends beyond the element.
`background-color`	Sets the background color of the element.
`background-image`	Specifies a background image for an element.
`background-origin`	Specifies the background image origin position.
`background-position`	Sets the position of a background image.
`background-repeat`	Specifies how the background image is repeated.
`background-size`	Sets the size of the background image.
`border`	Specifies a border for an element
`border-bottom`	Specifies a bottom border for an element.
`border-bottom-color`	Sets the color of a bottom border .
`border-bottom-left-radius`	Sets the border radius of the bottom left corner.
`border-bottom-right-radius`	Sets the border radius of the bottom right corner

`border-bottom-style`	Sets the style of the bottom border.
`border-bottom-width`	Sets the width of the bottom border
`border-collapse`	Sets table borders to single collapsed line or separated.
`border-color`	Sets the color of the border.
`border-image`	Defines an image as border, instead of a color.
`border-image-outset`	Sets how far a border image extends beyond the border.
`border-image-repeat`	Defines if and how the border image is repeated.
`border-image-slice`	Defines how the border image will be sliced.
`border-image-source`	Specifies the url of the border image file.
`border-image-width`	Sets the width of the image border.
`border-left`	Sets the left border of the element.
`border-left-color`	Sets the color of the left border.
`border-left-style`	Sets the style of the left border.
`border-left-width`	Sets the width of the left border.
`border-radius`	Sets the radius of the border.
`border-right`	Sets the right border of the element.
`border-right-color`	Sets the color of the right border.
`border-right-style`	Sets the style of the right border.
`border-right-width`	Sets the width of the right border.
`border-spacing`	Sets the adjacent table cell distance.
`border-style`	Defines the style of the border
`border-top`	Sets the top border of the element.
`border-top-color`	Sets the color of the top border.
`border-top-left-radius`	Sets the border radius of the top left corner.
`border-top-right-radius`	Sets the border radius of the top right corner.
`border-top-style`	Sets the style of the top border.
`border-top-width`	Sets the width of the top border.
`border-width`	Sets the border width of the element.

`bottom`	Positions the element from the bottom of the relative container.
`box-shadow`	Adds a shadow effect to an element.
`box-sizing`	Sets how element height and width are calculated.
`caption-side`	Defines on which side of the table a caption is placed.
`caret-color`	Sets the color of the blinking mouse caret.
`@charset`	Specifies the character encoding of the stylesheet.
`clear`	Sets the element side that does not allow floating elements.
`clip`	Sets how an image is cropped or clipped inside a container.
`clip-path`	Clips an element inside a specific shape or SVG.
`color`	Specifies the color of text in an element.
`column-count`	Divides an element into the specified number of columns.
`column-fill`	Specifies how divided columns are filled.
`column-gap`	Specifies the space between divided columns.
`column-rule`	Sets the style, width, and color of a column divider.
`column-rule-color`	Sets the color of a column divider.
`column-rule-style`	Sets the style of a column divider.
`column-rule-width`	Sets the width of a column divider.
`column-span`	Sets number of divided columns an element should span.
`column-width`	Specifies the width of a divided column.
`columns`	Divide an element into columns of a certain width.

`content`	Used to insert content before or after an element.
`counter-increment`	Increase or decrease a CSS counter.
`counter-reset`	Initialize or reset CSS counter.
`cursor`	Specifies the shape of the mouse cursor.
`direction`	Specifies the text writing direction of a block-level element.
`display`	Specify an element's display behavior.
`empty-cells`	Specifies whether empty table cell borders will be displayed.
`filter`	Adds an image enhancing effect to an image.
`flex`	Specifies the width of the flexible items.
`flex-basis`	Specifies the initial width of a flex item.
`flex-direction`	Specifies the direction for the flex item to align.
`flex-flow`	Controls the direction and wrapping of flexible items.
`flex-grow`	Specifies how a flex item can grow inside the container.
`flex-shrink`	Specifies how a flex item can shrink inside the container.
`flex-wrap`	Specifies how flexible items wrap inside the container.
`float`	Sets how an element is positioned relative to other elements.
`font`	Sets font family, variant, weight, height, and size for an element.
`@font-face`	Embeds a custom font inside a web page
`font-family`	Sets the font family for an element.
`font-kerning`	Sets the spacing between the font's characters.

`font-size`	Sets the size of the font for an element.
`font-size-adjust`	Specifies a fall-back font size.
`font-stretch`	Sets the text characters to a wider or narrower variant.
`font-style`	Set the font style to normal, italic, or oblique.
`font-variant`	Specifies that text is displayed in a small-caps font.
`font-weight`	Sets the weight or thickness of the font.
`grid`	Defines a grid layout with responsive rows and columns.
`grid-area`	Sets the size and location of grid items in a grid container.
`grid-auto-columns`	Specifies the size of the columns in a grid container.
`grid-auto-flow`	Specifies the initial placement of items in a grid container.
`grid-auto-rows`	Specifies the initial size of the items in a grid container.
`grid-column`	Specifies the size and location of a grid item in a grid container.
`grid-column-end`	Specifies in which column-line the grid item will end.
`grid-column-gap`	Specifies the gap size between columns in a grid container.
`grid-column-start`	Specifies in which column line the grid item will start.
`grid-gap`	Specifies the gap size between grid rows and columns.
`grid-row`	Specifies the grid item size and location in a grid container.
`grid-row-end`	Specifies in which row-line the grid item will end.

`grid-row-gap`	Specifies the gap size between rows in a grid container.
`grid-row-start`	Specifies in which row line the grid item will start
`grid-template`	Divides a page into sections with a size, position, and layer.
`grid-template-areas`	Specifies area in a grid container.
`grid-template-columns`	Sets the number and width of columns in a grid container.
`grid-template-rows`	Sets the number and height of rows in a grid container.
`height`	Sets the height of an element.
`hyphens`	Specifies hyphenation with wrap opportunities in a line of text.
`@import`	Imports a style sheet inside another style sheet.
`justify-content`	Defines the alignment of items in a flex container.
`@keyframes`	Defines the CSS style to animate.
`left`	Positions the element from the left of the relative container.
`letter-spacing`	Sets the spacing between characters.
`line-height`	Sets the vertical spacing between lines of text.
`list-style`	Defines the markers (bullet points) for items in a list.
`list-style-image`	Defines an image markers (bullet points) for items in a list.
`list-style-position`	Sets the marker (bullet point) positions for items in a list
`list-style-type`	Defines the marker types (bullet points) for items in a list
`margin`	Sets the margin (outside spacing) for an

	element.
`margin-bottom`	Sets the bottom margin (outside spacing) for an element.
`margin-left`	Sets the left margin (outside spacing) for an element.
`margin-right`	Sets the right margin (outside spacing) for an element.
`margin-top`	Sets the top margin (outside spacing) for an element.
`max-height`	Sets the maximum height for an element.
`max-width`	Sets the maximum width for an element.
`@media`	Applies media queries to a page.
`min-height`	Sets the minimum height for an element.
`min-width`	Sets the minimum width for an element.
`object-fit`	Specifies how an image or video fits inside a container.
`object-position`	Specifies the image or video position inside a container.
`opacity`	Sets the opacity (transparency) of the element.
`order`	Specifies the order of an item in a flex container.
`outline`	Adds an outline (highlighted border) to an element.
`outline-color`	Sets the color of an outline.
`outline-offset`	Sets the space between the outline and border.
`outline-style`	Sets the style of an outline.
`outline-width`	Sets the width of an outline.
`overflow`	Specifies the flow of content that exceeds the container.

`overflow-x`	Specifies the flow of content that exceeds the container width.
`overflow-y`	Specifies the flow of content that exceeds the container height.
`padding`	Sets the spacing between content and element border.
`padding-bottom`	Sets the spacing between content and bottom element border.
`padding-left`	Sets the spacing between content and left element border.
`padding-right`	Sets the spacing between content and right element border.
`padding-top`	Sets the spacing between content and top element border.
`page-break-after`	Adds a print page-break after an element.
`page-break-before`	Adds a print page-break before an element.
`page-break-inside`	Specifies if print page-break is allowed inside an element.
`perspective`	Adds perspective to a 3D-positioned element.
`perspective-origin`	Sets the origin of the perspective for a 3D-positioned element.
`pointer-events`	Specifies whether element reacts to pointer events or not.
`position`	Sets the element's positioning method.
`quotes`	Defines the quotation marks to be used on text.
`right`	Positions the element from the right of the relative container.

`scroll-behavior`	Specifies the scrolling behavior of an element
`table-layout`	Aligns elements according to a table with rows and columns.
`text-align`	Sets the alignment of text inside an element.
`text-align-last`	Sets the alignment for the last line of text.
`text-decoration`	Defines the style and color of underlined text.
`text-decoration-color`	Defines the color of underlined text.
`text-decoration-line`	Defines the kind of line to use with text.
`text-decoration-style`	Defines the style of underlined text.
`text-indent`	Sets the indentation to the beginning of text.
`text-justify`	Defines the text justification inside a container.
`text-overflow`	Sets the display behavior of text that overflows a container.
`text-shadow`	Adds a shadow effect to text.
`text-transform`	Defines text capitalization or casing.
`top`	Positions the element from the top of the relative container
`transform`	Applies a 2D or 3D transformation to an element.
`transform-origin`	Sets the origin for the transformation of the element.
`transform-style`	Specifies the display behavior of 3D space nested elements.
`transition`	Creates transitions from one property value to another.
`transition-delay`	Creates a delay before the transition effect starts.
`transition-duration`	Specifies the time the transition will take.
`transition-property`	Specifies the CSS property that will

	transition.
`transition-timing-function`	Defines the speed curve function of the transition.
`user-select`	Specifies how text can be selected (highlighted)
`vertical-align`	Specifies vertical alignment of an element.
`visibility`	Specifies the visibility of an element.
`white-space`	Specifies how white-space is handled inside an element.
`width`	Sets the width of an element.
`word-break`	Specifies how line breaks take place.
`word-spacing`	Sets the spacing between words.
`word-wrap`	Specifies how long words can be wrapped.
`writing-mode`	Sets the text reading orientation: top to bottom, etc.
`z-index`	Sets the vertical stacking order relative to other elements

Vendor Prefixes

Browser vendors regularly experiment with new, non-standard CSS properties. To indicate that these are browser specific they are prefixed, like so:

- `-webkit-...`
- `-moz-...`
- `-ms-...`
- `-o-...`

During the transition period when a property is supported by all browsers - but not fully standardized - you will see CSS like this:

```
-webkit-transition: background-color 2s ease-out;

-moz-transition: background-color 2s ease-out;
```

```
-ms-transition: background-color 2s ease-out;

-o-transition: background-color 2s ease-out;

transition: background-color 2s ease-out;
```

When they become standardized, the prefix becomes obsolete. Depending on the browsers your users use, you may decide to include prefixed properties.

16. CSS Colors

CSS colors are specified with predefined color names, or with RGB, HEX, HSL, RGBA, or HSLA values.

Color Names

In CSS, a color can be specified by using a color name:

Tomato
Orange
DodgerBlue
MediumSeaGreen
Gray
SlateBlue
Violet
LightGray

CSS Background Color

You can set the background color for HTML elements:

Hello World

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.

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

CSS Text Color

You can set the color of text:

Hello World

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

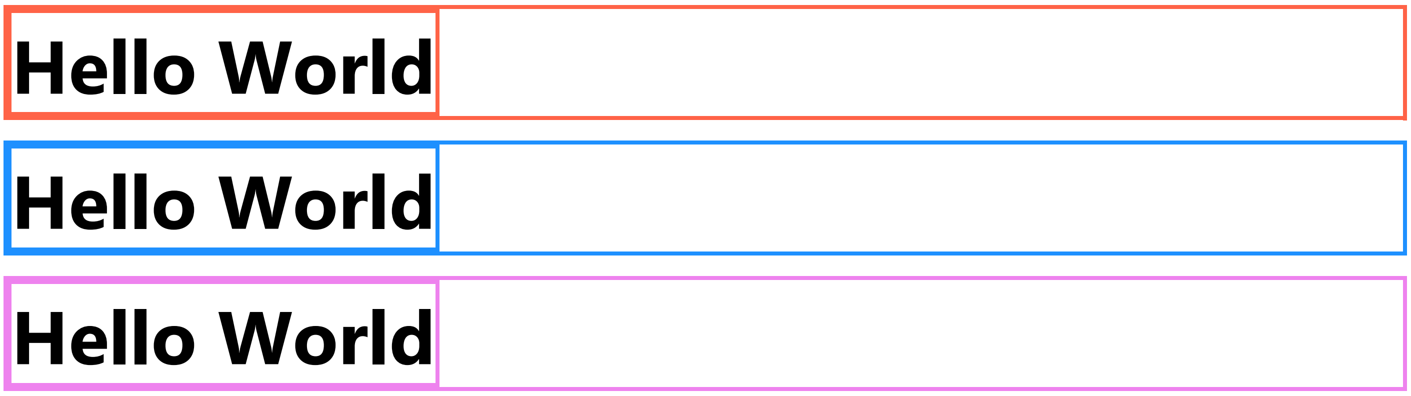
Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Example

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

CSS Border Color

You can set the color of borders:



Example

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>
```

CSS Color Values

In CSS, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values.

The following three <div> elements have their background color set with RGB, HEX, and HSL values:

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

The following two <div> elements have their background color set with RGBA and HSLA values, which add an Alpha channel to the color (here we have 50% transparency):

Example

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>
```

16(i). CSS RGB and RGBA Colors

An RGB color value represents RED, GREEN, and BLUE light sources.

An RGBA color value is an extension of RGB with an Alpha channel (opacity).

RGB Color Values

In HTML, a color can be specified as an RGB value, using this formula:

`rgb(red, green, blue)`

Each parameter (red, green, and blue) defines the intensity of the color with a value between 0 and 255.

This means that there are 256 x 256 x 256 = 16777216 possible colors!

For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255), and the other two (green and blue) are set to 0.

Another example, rgb(0, 255, 0) is displayed as green, because green is set to its highest value (255), and the other two (red and blue) are set to 0.

To display black, set all color parameters to 0, like this: rgb(0, 0, 0).

To display white, set all color parameters to 255, like this: rgb(255, 255, 255).

Experiment by mixing the RGB values below: **`rgb(255, 99, 71)`**

```
<h1 style="background-color:rgb(255, 0, 0);">rgb(255, 0, 0)</h1>
<h1 style="background-color:rgb(0, 0, 255);">rgb(0, 0, 255)</h1>
<h1 style="background-color:rgb(60, 179, 113);">rgb(60, 179,
113)</h1>
<h1 style="background-color:rgb(238, 130, 238);">rgb(238, 130,
238)</h1>
<h1 style="background-color:rgb(255, 165, 0);">rgb(255, 165, 0)</h1>
<h1 style="background-color:rgb(106, 90, 205);">rgb(106, 90,
205)</h1>
```

Shades of Gray

Shades of gray are often defined using equal values for all three parameters:

<h1 style="background-color:rgb(60, 60, 60);">rgb(60, 60, 60)</h1>

<h1 style="background-color:rgb(100, 100, 100);">rgb(100, 100, 100)</h1>

<h1 style="background-color:rgb(140, 140, 140);">rgb(140, 140, 140)</h1>

<h1 style="background-color:rgb(180, 180, 180);">rgb(180, 180, 180)</h1>

<h1 style="background-color:rgb(200, 200, 200);">rgb(200, 200, 200)</h1>

<h1 style="background-color:rgb(240, 240, 240);">rgb(240, 240, 240)</h1>

RGBA Color Values

RGBA color values are an extension of RGB color values with an Alpha channel - which specifies the opacity for a color.

An RGBA color value is specified with:

`rgba(red, green, blue, alpha)`

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

Experiment by mixing the RGBA values below:

<h1 style="background-color:rgba(255, 99, 71, 0);">rgba(255, 99, 71, 0)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.2);">rgba(255, 99, 71, 0.2)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.4);">rgba(255, 99, 71, 0.4)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.6);">rgba(255, 99, 71, 0.6)</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.8);">rgba(255, 99, 71, 0.8)</h1>

```
<h1 style="background-color:rgba(255, 99, 71, 1);">rgba(255, 99, 71,
1)</h1>
```

16(ii). CSS HEX Colors

A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color.

HEX Color Values

In HTML, a color can be specified using a hexadecimal value in the form:

#rrggbb

Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

For example, #ff0000 is displayed as red, because red is set to its highest value (ff), and the other two (green and blue) are set to 00.

Another example, #00ff00 is displayed as green, because green is set to its highest value (ff), and the other two (red and blue) are set to 00.

To display black, set all color parameters to 00, like this: #000000.

To display white, set all color parameters to ff, like this: #ffffff.

Experiment by mixing the HEX values below:

```
<h1 style="background-color:#ff0000;">#ff0000</h1>
<h1 style="background-color:#0000ff;">#0000ff</h1>
<h1 style="background-color:#3cb371;">#3cb371</h1>
<h1 style="background-color:#ee82ee;">#ee82ee</h1>
<h1 style="background-color:#ffa500;">#ffa500</h1>
<h1 style="background-color:#6a5acd;">#6a5acd</h1>
```

Shades of Gray

Shades of gray are often defined using equal values for all three parameters:

```
<h1 style="background-color:#404040;">#404040</h1>
<h1 style="background-color:#686868;">#686868</h1>
<h1 style="background-color:#a0a0a0;">#a0a0a0</h1>
<h1 style="background-color:#bebebe;">#bebebe</h1>
<h1 style="background-color:#dcdcdc;">#dcdcdc</h1>
<h1 style="background-color:#f8f8f8;">#f8f8f8</h1>
```

16(iii). HTML HSL and HSLA Colors

HSL stands for hue, saturation, and lightness.

HSLA color values are an extension of HSL with an Alpha channel (opacity).

HSL Color Values

In CSS, a color can be specified using hue, saturation, and lightness (HSL) in the form:

`hsl(hue, saturation, lightness)`

Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.

Saturation is a percentage value. 0% means a shade of gray, and 100% is the full color.

Lightness is also a percentage value. 0% is black, and 100% is white.

Experiment by mixing the HSL values below:

```
<h1 style="background-color:hsl(0, 100%, 50%);">hsl(0, 100%,
50%)</h1>
<h1 style="background-color:hsl(240, 100%, 50%);">hsl(240, 100%,
50%)</h1>
<h1 style="background-color:hsl(147, 50%, 47%);">hsl(147, 50%,
47%)</h1>
<h1 style="background-color:hsl(300, 76%, 72%);">hsl(300, 76%,
72%)</h1>
<h1 style="background-color:hsl(39, 100%, 50%);">hsl(39, 100%,
50%)</h1>
<h1 style="background-color:hsl(248, 53%, 58%);">hsl(248, 53%,
58%)</h1>
```

Saturation

Saturation can be described as the intensity of a color.

100% is pure color, no shades of gray.

50% is 50% gray, but you can still see the color.

0% is completely gray; you can no longer see the color.

```
<h1 style="background-color:hsl(0, 100%, 50%);">hsl(0,
100%, 50%)</h1>

<h1 style="background-color:hsl(0, 80%, 50%);">hsl(0, 80%,
50%)</h1>

<h1 style="background-color:hsl(0, 60%, 50%);">hsl(0, 60%,
50%)</h1>

<h1 style="background-color:hsl(0, 40%, 50%);">hsl(0, 40%,
50%)</h1>

<h1 style="background-color:hsl(0, 20%, 50%);">hsl(0, 20%,
50%)</h1>

<h1 style="background-color:hsl(0, 0%, 50%);">hsl(0, 0%,
50%)</h1>
```

```
<p>With HSL colors, less saturation mean less color. 0% is
completely gray.</p>
```

Lightness

The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light), and 100% means full lightness (white).

Example

```
<h1 style="background-color:hsl(0, 100%, 0%);">hsl(0, 100%, 0%)</h1>
<h1 style="background-color:hsl(0, 100%, 25%);">hsl(0, 100%,
25%)</h1>
<h1 style="background-color:hsl(0, 100%, 50%);">hsl(0, 100%,
50%)</h1>
<h1 style="background-color:hsl(0, 100%, 75%);">hsl(0, 100%,
75%)</h1>
<h1 style="background-color:hsl(0, 100%, 90%);">hsl(0, 100%,
90%)</h1>
<h1 style="background-color:hsl(0, 100%, 100%);">hsl(0, 100%,
100%)</h1>

<p>With HSL colors, 0% lightness means black, and 100 lightness
means white.</p>
```

Shades of Gray

Shades of gray are often defined by setting the hue and saturation to 0, and adjusting the lightness from 0% to 100% to get darker/lighter shades:

Example

```
<h1 style="background-color:hsl(0, 0%, 20%);">hsl(0, 0%, 20%)</h1>
<h1 style="background-color:hsl(0, 0%, 30%);">hsl(0, 0%, 30%)</h1>
<h1 style="background-color:hsl(0, 0%, 40%);">hsl(0, 0%, 40%)</h1>
<h1 style="background-color:hsl(0, 0%, 60%);">hsl(0, 0%, 60%)</h1>
<h1 style="background-color:hsl(0, 0%, 70%);">hsl(0, 0%, 70%)</h1>
<h1 style="background-color:hsl(0, 0%, 90%);">hsl(0, 0%, 90%)</h1>
```

HSLA Color Values

HSLA color values are an extension of HSL color values, with an Alpha channel - which specifies the opacity for a color.

An HSLA color value is specified with:

`hsla(hue, saturation, lightness, alpha)`

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

Experiment by mixing the HSLA values below:

```
<h1 style="background-color:hsla(9, 100%, 64%, 0);">hsla(9, 100%,
64%, 0)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.2);">hsla(9, 100%,
64%, 0.2)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.4);">hsla(9, 100%,
64%, 0.4)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.6);">hsla(9, 100%,
64%, 0.6)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.8);">hsla(9, 100%,
64%, 0.8)</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 1);">hsla(9, 100%,
64%, 1)</h1>
```

17. CSS Backgrounds

The CSS background properties are used to add background effects for elements.

In these chapters, you will learn about the following CSS background properties:

- `background-color`
- `background-image`
- `background-repeat`
- `background-attachment`
- `background-position`
- `background` (shorthand property)

CSS background-color

The `background-color` property specifies the background color of an element.

Example

The background color of a page is set like this:

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

With CSS, a color is most often specified by:

- a valid color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

Other Elements

You can set the background color for any HTML elements:

Example

Here, the <h1>, <p>, and <div> elements will have different background colors:

```
h1 {
  background-color: green;
}

div {
  background-color: lightblue;
}

p {
  background-color: yellow;
}
```

Opacity / Transparency

The `opacity` property specifies the opacity/transparency of an element. It can take a value from 0.0 - 1.0. The lower value, the more transparent:

Example

```
div {
  background-color: green;
  opacity: 0.3;
}
```

Note: When using the `opacity` property to add transparency to the background of an element, all of its child elements inherit the same transparency. This can make the text inside a fully transparent element hard to read.

Transparency using RGBA

If you do not want to apply opacity to child elements, like in our example above, use **RGBA** color values. The following example sets the opacity for the background color and not the text:

You learned from our CSS Colors Chapter, that you can use RGB as a color value. In addition to RGB, you can use an RGB color value with an **alpha** channel (RGB**A**) - which specifies the opacity for a color.

An RGBA color value is specified with: rgba(red, green, blue, alpha). The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

Example

```
div {
  background: rgba(0, 128, 0, 0.3) /* Green background with 30%
opacity */
}
```

The CSS Background Color Property

Property	Description
background-color	Sets the background color of an element

17(i). CSS Background Image

CSS background-image

The `background-image` property specifies an image to use as the background of an element.

By default, the image is repeated so it covers the entire element.

Example

Set the background image for a page:

```
body {
  background-image: url("paper.gif");
}
```

Example

This example shows a **bad combination** of text and background image. The text is hardly readable:

```
body {
  background-image: url("bgdesert.jpg");
}
```

Note: When using a background image, use an image that does not disturb the text.

The background image can also be set for specific elements, like the <p> element:

Example

```
p {
  background-image: url("paper.gif");
}
```

The CSS Background Image Property

Property	Description
background-image	Sets the background image for an element

17(ii). CSS Background Image Repeat

CSS background-repeat

By default, the `background-image` property repeats an image both horizontally and vertically.

Some images should be repeated only horizontally or vertically, or they will look strange, like this:

Example

```
body {
  background-image: url("gradient_bg.png");
}
```

If the image above is repeated only horizontally (`background-repeat: repeat-x;`), the background will look better:

Example

```
body {
  background-image: url("gradient_bg.png");
  background-repeat: repeat-x;
}
```

Tip: To repeat an image vertically, set `background-repeat: repeat-y;`

CSS background-repeat: no-repeat

Showing the background image only once is also specified by the `background-repeat` property:

Example

Show the background image only once:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
}
```

In the example above, the background image is placed in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much.

CSS background-position

The `background-position` property is used to specify the position of the background image.

Example

Position the background image in the top-right corner:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

The CSS Background Repeat and Position Properties

Property	Description
background-position	Sets the starting position of a background image
background-repeat	Sets how a background image will be repeated

17(iii). CSS Background Attachment

CSS background-attachment

The `background-attachment` property specifies whether the background image should scroll or be fixed (will not scroll with the rest of the page):

Example

Specify that the background image should be fixed:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: fixed;
}
```

Example

Specify that the background image should scroll with the rest of the page:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
  background-attachment: scroll;
}
```

The CSS Background Attachment Property

Property	Description
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page

17(iv)CSS Background Shorthand

CSS background - Shorthand property

To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property.

Instead of writing:

```
body {
  background-color: #ffffff;
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

You can use the shorthand property `background`:

Example

Use the shorthand property to set the background properties in one declaration:

```
body {
  background: #ffffff url("img_tree.png") no-repeat right top;
}
```

When using the shorthand property the order of the property values is:

- `background-color`
- `background-image`
- `background-repeat`
- `background-attachment`
- `background-position`

It does not matter if one of the property values is missing, as long as the other ones are in this order. Note that we do not use the background-attachment property in the examples above, as it does not have a value.

Exercise:

Set the background color of the <h1> element to "lightblue".

```
<style>

h1 {

         : lightblue;

}

</style>

<body>
```

```
<h1>This is a heading</h1>

<p>This is a heading</p>

<p>This is a heading</p>

</body>
```

All CSS Background Properties

Property	Description
background	Sets all the background properties in one declaration
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page
background-clip	Specifies the painting area of the background
background-color	Sets the background color of an element
background-image	Sets the background image for an element
background-origin	Specifies where the background image(s) is/are positioned
background-position	Sets the starting position of a background image
background-repeat	Sets how a background image will be repeated
background-size	Specifies the size of the background image(s)

Week 3:

Week 3-4: Structuring Content

HTML Lists

HTML lists allow web developers to group a set of related items in lists.

Example

An unordered HTML list:

- Item
- Item
- Item
- Item

An ordered HTML list:

1. First item
2. Second item
3. Third item
4. Fourth item

Unordered HTML List

An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with bullets (small black circles) by default:

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Ordered HTML List

An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with numbers by default:

Example

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

HTML Description Lists

HTML also supports description lists.

A description list is a list of terms, with a description of each term.

The `<dl>` tag defines the description list, the `<dt>` tag defines the term (name), and the `<dd>` tag describes each term:

Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

HTML List Tags

Tag	Description
<ul>	Defines an unordered list
<ol>	Defines an ordered list
<li>	Defines a list item
<dl>	Defines a description list
<dt>	Defines a term in a description list

<dd>	Describes the term in a description list

HTML Unordered Lists

The HTML `<ul>` tag defines an unordered (bulleted) list.

Unordered HTML List

An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with bullets (small black circles) by default:

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Unordered HTML List - Choose List Item Marker

The CSS `list-style-type` property is used to define the style of the list item marker. It can have one of the following values:

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

Example - Disc

```
<ul style="list-style-type:disc;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Circle

```
<ul style="list-style-type:circle;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Square

```
<ul style="list-style-type:square;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - None

```
<ul style="list-style-type:none;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Nested HTML Lists

Lists can be nested (list inside list):

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

Note: A list item (`<li>`) can contain a new list, and other HTML elements, like images and links, etc.

Horizontal List with CSS

HTML lists can be styled in many different ways with CSS.

One popular way is to style a list horizontally, to create a navigation menu:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333333;
}

li {
  float: left;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 16px;
  text-decoration: none;
}

li a:hover {
  background-color: #111111;
}
```

```
</style>
</head>
<body>

<ul>
  <li><a href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li><a href="#contact">Contact</a></li>
  <li><a href="#about">About</a></li>
</ul>

</body>
</html>
```

Chapter Summary

- Use the HTML `<ul>` element to define an unordered list
- Use the CSS `list-style-type` property to define the list item marker
- Use the HTML `<li>` element to define a list item
- Lists can be nested
- List items can contain other HTML elements
- Use the CSS property `float:left` to display a list horizontally

HTML List Tags

Tag	Description
<ul>	Defines an unordered list
<ol>	Defines an ordered list
<li>	Defines a list item
<dl>	Defines a description list
<dt>	Defines a term in a description list
<dd>	Describes the term in a description list

HTML Ordered Lists

The HTML `<ol>` tag defines an ordered list. An ordered list can be numerical or alphabetical.

Ordered HTML List

An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with numbers by default:

Example

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Ordered HTML List - The Type Attribute

The `type` attribute of the `<ol>` tag, defines the type of the list item marker:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers

type="i"	The list items will be numbered with lowercase roman numbers

Numbers:

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Letters:

```
<ol type="A">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Letters:

```
<ol type="a">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Roman Numbers:

```
<ol type="I">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Roman Numbers:

```
<ol type="i">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Control List Counting

By default, an ordered list will start counting from 1. If you want to start counting from a specified number, you can use the `start` attribute:

Example

```
<ol start="50">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Nested HTML Lists

Lists can be nested (list inside list):

Example

```
<ol>
  <li>Coffee</li>
  <li>Tea
    <ol>
      <li>Black tea</li>
      <li>Green tea</li>
    </ol>
  </li>
  <li>Milk</li>
</ol>
```

Note: A list item (`<li>`) can contain a new list, and other HTML elements, like images and links, etc.

Chapter Summary

- Use the HTML `<ol>` element to define an ordered list
- Use the HTML `type` attribute to define the numbering type
- Use the HTML `<li>` element to define a list item
- Lists can be nested
- List items can contain other HTML elements

HTML List Tags

Tag	Description
<ul>	Defines an unordered list
<ol>	Defines an ordered list
<li>	Defines a list item
<dl>	Defines a description list
<dt>	Defines a term in a description list
<dd>	Describes the term in a description list

HTML Other Lists

HTML also supports description lists.

HTML Description Lists

A description list is a list of terms, with a description of each term.

The <dl> tag defines the description list, the <dt> tag defines the term (name), and the <dd> tag describes each term:

Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Chapter Summary

- Use the HTML `<dl>` element to define a description list
- Use the HTML `<dt>` element to define the description term
- Use the HTML `<dd>` element to describe the term in a description list

---

HTML Exercises

Exercise:

Add a list item with the text "Coffee" inside the `<ul>` element.

```
<ul>   Coffee   </ul>
```

HTML Tables

HTML tables allow web developers to arrange data into rows and columns.

Example

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

Define an HTML Table

A table in HTML consists of table cells inside rows and columns.

Example

A simple HTML table:

```
<table>
  <tr>
    <th>Company</th>
    <th>Contact</th>
    <th>Country</th>
  </tr>
  <tr>
    <td>Alfreds Futterkiste</td>
    <td>Maria Anders</td>
    <td>Germany</td>
  </tr>
  <tr>
    <td>Centro comercial Moctezuma</td>
    <td>Francisco Chang</td>
    <td>Mexico</td>
  </tr>
</table>
```

Table Cells

Each table cell is defined by a `<td>` and a `</td>` tag.

`td` stands for table data.

Everything between `<td>` and `</td>` are the content of the table cell.

Example

```
<table>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
    <td>Linus</td>
  </tr>
</table>
```

Note: A table cell can contain all sorts of HTML elements: text, images, lists, links, other tables, etc.

Table Rows

Each table row starts with a `<tr>` and ends with a `</tr>` tag.

`tr` stands for table row.

Example

```
<table>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
    <td>Linus</td>
  </tr>
  <tr>
    <td>16</td>
    <td>14</td>
    <td>10</td>
  </tr>
</table>
```

You can have as many rows as you like in a table; just make sure that the number of cells are the same in each row.

Note: There are times when a row can have less or more cells than another. You will learn about that in a later chapter.

Table Headers

Sometimes you want your cells to be table header cells. In those cases use the `<th>` tag instead of the `<td>` tag:

`th` stands for table header.

Example

Let the first row be table header cells:

```
<table>
  <tr>
    <th>Person 1</th>
    <th>Person 2</th>
    <th>Person 3</th>
  </tr>
  <tr>
    <td>Emil</td>
    <td>Tobias</td>
    <td>Linus</td>
  </tr>
  <tr>
    <td>16</td>
    <td>14</td>
    <td>10</td>
  </tr>
</table>
```

By default, the text in `<th>` elements are bold and centered, but you can change that with CSS.

Exercise:

Add a table row with two table headers.

The two table headers should have the value "Name" and "Age".

```
<table>




  <tr>
    <td>Jill Smith</td>
    <td>50</td>
  </tr>
</table>
```

HTML Table Tags

Tag	Description
<table>	Defines a table
<th>	Defines a header cell in a table
<tr>	Defines a row in a table
<td>	Defines a cell in a table
<caption>	Defines a table caption
<colgroup>	Specifies a group of one or more columns in a table for formatting
<col>	Specifies column properties for each column within a <colgroup> element
<thead>	Groups the header content in a table
<tbody>	Groups the body content in a table
<tfoot>	Groups the footer content in a table

HTML Table Borders

HTML tables can have borders of different styles and shapes.

How To Add a Border

To add a border, use the CSS `border` property on `table`, `th`, and `td` elements:

Example

```css
table, th, td {
  border: 1px solid black;
}
```

Collapsed Table Borders

To avoid having double borders like in the example above, set the CSS `border-collapse` property to `collapse`.

This will make the borders collapse into a single border:

Example

```css
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
```

Style Table Borders

If you set a background color of each cell, and give the border a white color (the same as the document background), you get the impression of an invisible border:

Example

```css
table, th, td {
  border: 1px solid white;
  border-collapse: collapse;
}
th, td {
  background-color: #96D4D4;
}
```

Round Table Borders

With the `border-radius` property, the borders get rounded corners:

Example

```css
table, th, td {
  border: 1px solid black;
  border-radius: 10px;
}
```

Skip the border around the table by leaving out `table` from the css selector:

Example

```
th, td {
  border: 1px solid black;
  border-radius: 10px;
}
```

Dotted Table Borders

With the `border-style` property, you can set the appearance of the border.

The following values are allowed:

- `dotted`
- `dashed`
- `solid`
- `double`
- `groove`
- `ridge`
- `inset`
- `outset`
- `none`
- `hidden`

Example

```
th, td {
  border-style: dotted;
}
```

Border Color

With the `border-color` property, you can set the color of the border.

Example

```
 th, td {
  border-color: #96D4D4;
}
```

HTML Table Sizes

HTML tables can have different sizes for each column, row or the entire table.

Use the `style` attribute with the `width` or `height` properties to specify the size of a table, row or column.

HTML Table Width

To set the width of a table, add the `style` attribute to the `<table>` element:

Example

Set the width of the table to 100%:

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Note: Using a percentage as the size unit for a width means how wide will this element be compared to its parent element, which in this case is the `<body>` element.

HTML Table Column Width

To set the size of a specific column, add the `style` attribute on a `<th>` or `<td>` element:

Example

Set the width of the first column to 70%:

```
<table style="width:100%">
  <tr>
    <th style="width:70%">Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

HTML Table Row Height

To set the height of a specific row, add the `style` attribute on a table row element:

Example

Set the height of the second row to 200 pixels:

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr style="height:200px">
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Exercise:

Use CSS styles to make the table 300 pixels wide.

```
<table >
  <tr>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Points</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
</table>
```

HTML Table Headers

HTML tables can have headers for each column or row, or for many columns/rows.

HTML Table Headers

Table headers are defined with `th` elements. Each `th` element represents a table cell.

Example

```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Vertical Table Headers

To use the first column as table headers, define the first cell in each row as a `<th>` element:

Example

```
<table>
  <tr>
    <th>Firstname</th>
    <td>Jill</td>
    <td>Eve</td>
  </tr>
  <tr>
    <th>Lastname</th>
    <td>Smith</td>
    <td>Jackson</td>
  </tr>
  <tr>
    <th>Age</th>
    <td>94</td>
    <td>50</td>
  </tr>
</table>
```

Align Table Headers

By default, table headers are bold and centered:

Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94

To left-align the table headers, use the CSS `text-align` property:

Example

```
th {
  text-align: left;
}
```

Header for Multiple Columns

You can have a header that spans over two or more columns.

Name		Age
Jill	Smith	50
Eve	Jackson	94

To do this, use the `colspan` attribute on the `<th>` element:

Example

```
<style>
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
</style>
<table>
  <tr>
```

```
    <th colspan="2">Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Table Caption

You can add a caption that serves as a heading for the entire table.

Monthly savings

Month	Savings
January	$100
February	$50

To add a caption to a table, use the `<caption>` tag:

```
<style>

table, th, td {

  border: 1px solid black;

  border-collapse: collapse;

}

th, td {

  padding: 5px;

  text-align: left;

}

</style>
```

```
<table style="width:100%">
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$50</td>
  </tr>
</table>
```

Note: The <caption> tag should be inserted immediately after the <table> tag.

Exercise:

Add a table caption that says "Names".

```
<table>

  <tr>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Points</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
</table>
```

HTML Table Padding & Spacing

HTML tables can adjust the padding inside the cells, and also the space between the cells.

With Padding

hello	hello	hello
hello	hello	hello
hello	hello	hello

With Spacing

hello	hello	hello
hello	hello	hello
hello	hello	hello

HTML Table - Cell Padding

Cell padding is the space between the cell edges and the cell content.

By default the padding is set to 0.

To add padding on table cells, use the CSS `padding` property:

Example

```css
th, td {
  padding: 15px;
}
```

To add padding only above the content, use the `padding-top` property.

And the others sides with the `padding-bottom`, `padding-left`, and `padding-right` properties:

Example

```
th, td {
  padding-top: 10px;
  padding-bottom: 20px;
  padding-left: 30px;
  padding-right: 40px;
}
```

HTML Table - Cell Spacing

Cell spacing is the space between each cell.

By default the space is set to 2 pixels.

To change the space between table cells, use the CSS `border-spacing` property on the `table` element:

Example

```
table {
  border-spacing: 30px;
}
```

HTML Table Colspan & Rowspan

HTML tables can have cells that span over multiple rows and/or columns.

HTML Table - Colspan

To make a cell span over multiple columns, use the `colspan` attribute:

Example

```
<table>
  <tr>
    <th colspan="2">Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>43</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>57</td>
  </tr>
</table>
```

Note: The value of the `colspan` attribute represents the number of columns to span.

HTML Table - Rowspan

To make a cell span over multiple rows, use the `rowspan` attribute:

Example

```
<style>
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
</style>
<table>
  <tr>
    <th>Name</th>
    <td>Jill</td>
  </tr>
  <tr>
    <th rowspan="2">Phone</th>
    <td>555-1234</td>
  </tr>
  <tr>
    <td>555-8745</td>
</tr>
</table>
```

Note: The value of the `rowspan` attribute represents the number of rows to span.

Exercise:

Use the correct HTML attribute to make the first TH element span two columns.

```
<table>
  <tr>
    <th >Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

HTML Table Styling

Use CSS to make your tables look better.

HTML Table - Zebra Stripes

If you add a background color on every other table row, you will get a nice zebra stripes effect.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

To style every other table row element, use the `:nth-child(even)` selector like this:

Example

```
tr:nth-child(even) {
  background-color: #D6EEEE;
}
```

Note: If you use `(odd)` instead of `(even)`, the styling will occur on row 1,3,5 etc. instead of 2,4,6 etc.

HTML Table - Vertical Zebra Stripes

To make vertical zebra stripes, style every other column, instead of every other row.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

Set the `:nth-child(even)` for table data elements like this:

Example

```
td:nth-child(even), th:nth-child(even) {
  background-color: #D6EEEE;
}
```

Note: Put the `:nth-child()` selector on both `th` and `td` elements if you want to have the styling on both headers and regular table cells.

Combine Vertical and Horizontal Zebra Stripes

You can combine the styling from the two examples above and you will have stripes on every other row and every other column.

If you use a transparent color you will get an overlapping effect.

Use an `rgba()` color to specify the transparency of the color:

Example

```
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}

tr:nth-child(even) {
  background-color: rgba(150, 212, 212, 0.4);
}

th:nth-child(even),td:nth-child(even) {
  background-color: rgba(150, 212, 212, 0.4);
}
```

```
<h2>Striped Table</h2>
<p>For zebra-striped tables, use the nth-child() selector and add a
background-color to all even (or odd) table rows:</p>

<table style="width:100%">
  <tr>
    <th>MON</th>
    <th>TUE</th>
    <th>WED</th>
    <th>THU</th>
    <th>FRI</th>
    <th>SAT</th>
    <th>SUN</th>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
</table>
```

Horizontal Dividers

First Name	Last Name	Savings

First Name	Last Name	Savings
Peter	Griffin	$100
Lois	Griffin	$150
Joe	Swanson	$300

If you specify borders only at the bottom of each table row, you will have a table with horizontal dividers.

Add the `border-bottom` property to all `tr` elements to get horizontal dividers:

Example

```
<style>
table {
  border-collapse: collapse;
  width: 100%;
}

tr {
  border-bottom: 1px solid #ddd;
}
</style>
</head>
<body>

<h2>Bordered Table Dividers</h2>
<p>Add the border-bottom property to the tr elements for horizontal
dividers:</p>

<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
  <th>Savings</th>
  </tr>
  <tr>
    <td>Peter</td>
    <td>Griffin</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>Lois</td>
    <td>Griffin</td>
    <td>$150</td>
  </tr>
  <tr>
    <td>Joe</td>
    <td>Swanson</td>
    <td>$300</td>
  </tr>
  <tr>
    <td>Cleveland</td>
    <td>Brown</td>
    <td>$250</td>
  </tr>
</table>
```

Hoverable Table

Use the `:hover` selector on `tr` to highlight table rows on mouse over:

```
tr:hover {background-color: #D6EEEE;}
```

HTML Table Colgroup

The `<colgroup>` element is used to style specific columns of a table.

HTML Table Colgroup

If you want to style the two first columns of a table, use the `<colgroup>` and `<col>` elements.

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

The `<colgroup>` element should be used as a container for the column specifications.

Each group is specified with a `<col>` element.

The `span` attribute specifies how many columns that get the style.

The `style` attribute specifies the style to give the columns.

Note: There is a very limited selection of legal CSS properties for colgroups.

Example

```
<style>
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
</style>
</head>
<body>

<h2>Colgroup</h2>
<p>Add the a colgroup with a col element that spans over two
columns to define a style for the two columns:</p>

<table style="width: 100%;">
<colgroup>
  <col span="2" style="background-color: #D6EEEE">
</colgroup>
<tr>
<th>MON</th>
<th>TUE</th>
<th>WED</th>
<th>THU</th>
<th>FRI</th>
<th>SAT</th>
<th>SUN</th>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
```

```
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
</table>
```

Note: The `<colgroup>` tag must be a child of a `<table>` element and should be placed before any other table elements, like `<thead>`, `<tr>`, `<td>` etc., but after the `<caption>` element, if present.

Legal CSS Properties

There is only a very limited selection of CSS properties that are allowed to be used in the colgroup:

`width` property
`visibility` property
`background` properties
`border` properties

All other CSS properties will have no effect on your tables.

Multiple Col Elements

If you want to style more columns with different styles, use more `<col>` elements inside the `<colgroup>`:

```
<style>

table, th, td {

  border: 1px solid black;

  border-collapse: collapse;

}

</style>

</head>

<body>


<h2>Multiple Col Elements</h2>

<p>Add multiple col elements in the colgroup:</p>


<table style="width: 100%;">

  <colgroup>

    <col span="2" style="background-color: #D6EEEE">

    <col span="3" style="background-color: pink">

  </colgroup>

<tr>

<th>MON</th>
```

```
<th>TUE</th>

<th>WED</th>

<th>THU</th>

<th>FRI</th>

<th>SAT</th>

<th>SUN</th>

</tr>

<tr>

<td>1</td>

<td>2</td>

<td>3</td>

<td>4</td>

<td>5</td>

<td>6</td>

<td>7</td>

</tr>

<tr>

<td>8</td>

<td>9</td>

<td>10</td>

<td>11</td>

<td>12</td>

<td>13</td>

<td>14</td>

</tr>

<tr>
```

```
<td>15</td>

<td>16</td>

<td>17</td>

<td>18</td>

<td>19</td>

<td>20</td>

<td>21</td>

</tr>

<tr>

<td>22</td>

<td>23</td>

<td>24</td>

<td>25</td>

<td>26</td>

<td>27</td>

<td>28</td>

</tr>

</table>
```

Empty Colgroups

If you want to style columns in the middle of a table, insert a "empty" `<col>` element (with no styles) for the columns before:

Example

```
<style>

table, th, td {
  border: 1px solid black;

  border-collapse: collapse;

}

</style>

</head>

<body>


<h2>Empty Colgroups</h2>

<p>Add "empty" col elements that represents the columns before the columns you want to style:</p>


<table style="width: 100%;">

<colgroup>

  <col span="3">

  <col span="2" style="background-color: pink">

</colgroup>

<tr>

<th>MON</th>

<th>TUE</th>

<th>WED</th>

<th>THU</th>

<th>FRI</th>

<th>SAT</th>

<th>SUN</th>

</tr>

<tr>
```

```
<td>1</td>

<td>2</td>

<td>3</td>

<td>4</td>

<td>5</td>

<td>6</td>

<td>7</td>

</tr>

<tr>

<td>8</td>

<td>9</td>

<td>10</td>

<td>11</td>

<td>12</td>

<td>13</td>

<td>14</td>

</tr>

<tr>

<td>15</td>

<td>16</td>

<td>17</td>

<td>18</td>

<td>19</td>

<td>20</td>

<td>21</td>

</tr>
```

```
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
</table>
```

Hide Columns

You can hide columns with the `visibility: collapse` property:

```
<style>
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
</style>
</head>
<body>

<h2>Hide Columns</h2>
<p>You can hide specific columns with the visibility property:</p>
```

```
<table style="width: 100%;">
<colgroup>
    <col span="2">
    <col span="3" style="visibility: collapse">
  </colgroup>
<tr>
<th>MON</th>
<th>TUE</th>
<th>WED</th>
<th>THU</th>
<th>FRI</th>
<th>SAT</th>
<th>SUN</th>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
```

```
<td>10</td>

<td>11</td>

<td>12</td>

<td>13</td>

<td>14</td>

</tr>

<tr>

<td>15</td>

<td>16</td>

<td>17</td>

<td>18</td>

<td>19</td>

<td>20</td>

<td>21</td>

</tr>

<tr>

<td>22</td>

<td>23</td>

<td>24</td>

<td>25</td>

<td>26</td>

<td>27</td>

<td>28</td>

</tr>

</table>
```

<p><b>Note:</b> The table columns does not collapse properly in Safari browsers.</p>

</body>

HTML Block and Inline Elements

Every HTML element has a default display value, depending on what type of element it is.

There are two display values: block and inline.

Block-level Elements

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are: `<p>` and `<div>`.

The `<p>` element defines a paragraph in an HTML document.

The `<div>` element defines a division or a section in an HTML document.

The <p> element is a block-level element.

The <div> element is a block-level element.

Example

```
<p>Hello World</p>
<div>Hello World</div>
```

Here are the block-level elements in HTML:

`<address>`

`<article>`

`<aside>`

`<blockquote>`

`<canvas>`

`<dd>`

`<div>`

`<dl>`

`<dt>`

`<fieldset>`

`<figcaption>`

`<figure>`

`<footer>`

`<form>`

`<h1>-<h6>`

`<header>`

`<hr>`

`<li>`

`<main>`

`<nav>`

`<noscript>`

`<ol>`

`<p>`

`<pre>`

`<section>`

`<table>`

`<tfoot>`

`<ul>`

`<video>`

Inline Elements

An inline element does not start on a new line.

An inline element only takes up as much width as necessary.

This is a <span> element inside a paragraph.

Example

```
<span>Hello World</span>
```

Here are the inline elements in HTML:

```
<a>

<abbr>

<acronym>

<b>

<bdo>

<big>

<br>

<button>

<cite>

<code>

<dfn>

<em>

<i>

<img>

<input>

<kbd>
```

`<label>`

`<map>`

`<object>`

`<output>`

`<q>`

`<samp>`

`<script>`

`<select>`

`<small>`

`<span>`

`<strong>`

`<sub>`

`<sup>`

`<textarea>`

`<time>`

`<tt>`

`<var>`

Note: An inline element cannot contain a block-level element!

The <div> Element

The `<div>` element is often used as a container for other HTML elements.

The `<div>` element has no required attributes, but `style`, `class` and `id` are common.

When used together with CSS, the `<div>` element can be used to style blocks of content:

Example

```
<div style="background-color:black;color:white;padding:20px;">
  <h2>London</h2>
  <p>London is the capital city of England. It is the most populous
city in the United Kingdom, with a metropolitan area of over 13
million inhabitants.</p>
</div>
```

The <span> Element

The `<span>` element is an inline container used to mark up a part of a text, or a part of a document.

The `<span>` element has no required attributes,
but `style`, `class` and `id` are common.

When used together with CSS, the `<span>` element can be used to style parts of the text:

Example

```
<p>My mother has <span style="color:blue;font-
weight:bold;">blue</span> eyes and my father
has <span style="color:darkolivegreen;font-weight:bold;">dark
green</span> eyes.</p>
```

Chapter Summary

- There are two display values: block and inline
- A block-level element always starts on a new line and takes up the full width available
- An inline element does not start on a new line and it only takes up as much width as necessary
- The `<div>` element is a block-level and is often used as a container for other HTML elements
- The `<span>` element is an inline container used to mark up a part of a text, or a part of a document

HTML Tags

Tag	Description
<div>	Defines a section in a document (block-level)

<span>	Defines a section in a document (inline)

CSS Box Model

All HTML elements can be considered as boxes.

The CSS Box Model

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Explanation of the different parts:

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

Example

Demonstration of the box model:

```
div {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```

Width and Height of an Element

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

Important: When you set the width and height properties of an element with CSS, you just set the width and height of the **content area**. To calculate the full size of an element, you must also add padding, borders and margins.

Example

This <div> element will have a total width of 350px:

```
div {
  width: 320px;
  padding: 10px;
  border: 5px solid gray;
  margin: 0;
}
```

Here is the calculation:

320px (width)
+ 20px (left + right padding)
+ 10px (left + right border)
+ 0px (left + right margin)
= 350px

The total width of an element should be calculated like this:

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

Exercise:

Set the width of the <div> element to "200px".

```
<style> {
  : ;
}
</style>

<body>

<div>
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
</div>

</body>
```

CSS Website Layout

Website Layout

A website is often divided into headers, menus, content and a footer:

Header

Navigation Menu

Content
Main Content
Content

Footer

There are tons of different layout designs to choose from. However, the structure above, is one of the most common, and we will take a closer look at it in this tutorial.

Header

A header is usually located at the top of the website (or right below a top navigation menu). It often contains a logo or the website name:

Example

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>CSS Website Layout</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-
scale=1">

<style>

body {

  margin: 0;

}


/* Style the header */

.header {

  background-color: #f1f1f1;

  padding: 20px;

  text-align: center;

}

</style>

</head>

<body>


<div class="header">

  <h1>Header</h1>

</div>
```

```
</body>

</html>
```

Navigation Bar

A navigation bar contains a list of links to help visitors navigating through your website:

Example

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>CSS Website Layout</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<style>

* {

  box-sizing: border-box;

}


body {

  margin: 0;

}


/* Style the header */

.header {

  background-color: #f1f1f1;

  padding: 20px;

  text-align: center;
}


/* Style the top navigation bar */

.topnav {

  overflow: hidden;

  background-color: #333;

}


/* Style the topnav links */

.topnav a {

  float: left;

  display: block;

  color: #f2f2f2;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}


/* Change color on hover */

.topnav a:hover {

  background-color: #ddd;

  color: black;

}

</style>

</head>

<body>


<div class="header">

  <h1>Header</h1>

</div>
```

```
<div class="topnav">

  <a href="#">Link</a>

  <a href="#">Link</a>

  <a href="#">Link</a>

</div>



</body>

</html>
```

Content

The layout in this section, often depends on the target users. The most common layout is one (or combining them) of the following:

- **1-column** (often used for mobile browsers)
- **2-column** (often used for tablets and laptops)
- **3-column layout** (only used for desktops)

1-column:

2-column:

3-column:

We will create a 3-column layout, and change it to a 1-column layout on smaller screens:

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>CSS Website Layout</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
* {
  box-sizing: border-box;
}

body {
  margin: 0;
}

/* Style the header */
.header {
  background-color: #f1f1f1;
  padding: 20px;
  text-align: center;
}

/* Style the top navigation bar */
.topnav {
  overflow: hidden;
  background-color: #333;
}

/* Style the topnav links */
.topnav a {
  float: left;
  display: block;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

/* Change color on hover */
.topnav a:hover {
  background-color: #ddd;
  color: black;
}

/* Create three equal columns that floats next to each other */
.column {
  float: left;
  width: 33.33%;
  padding: 15px;
}

/* Clear floats after the columns */
.row::after {
  content: "";
  display: table;
  clear: both;
}

/* Responsive layout - makes the three columns stack on top of each other instead of next to each other */
@media screen and (max-width:600px) {
  .column {
    width: 100%;
  }
}
</style>
```

```
</head>
<body>

<div class="header">
  <h1>Header</h1>
  <p>Resize the browser window to see the responsive effect.</p>
</div>

<div class="topnav">
  <a href="#">Link</a>
  <a href="#">Link</a>
  <a href="#">Link</a>
</div>

<div class="row">
  <div class="column">
    <h2>Column</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque ultricies, eget elementum magna tristique. Quisque vehicula, risus eget aliquam placerat, purus leo tincidunt eros, eget luctus quam orci in velit. Praesent scelerisque tortor sed accumsan convallis.</p>
  </div>

  <div class="column">
    <h2>Column</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque ultricies, eget elementum magna tristique. Quisque vehicula, risus eget aliquam placerat, purus leo tincidunt eros, eget luctus quam orci in velit. Praesent scelerisque tortor sed accumsan convallis.</p>
  </div>

  <div class="column">
    <h2>Column</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque ultricies, eget elementum magna tristique. Quisque vehicula, risus eget aliquam placerat, purus leo tincidunt eros, eget luctus quam orci in velit. Praesent scelerisque tortor sed accumsan convallis.</p>
  </div>
</div>

</body>
</html>
```

Tip: To create a 2-column layout, change the width to 50%. To create a 4-column layout, use 25%, etc.

Unequal Columns

The main content is the biggest and the most important part of your site.

It is common with **unequal** column widths, so that most of the space is reserved for the main content. The side content (if any) is often used as an alternative navigation or to specify information relevant to the main content. Change the widths as you like, only remember that it should add up to 100% in total:

Example

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>CSS Website Layout</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-
scale=1">

<style>

* {

  box-sizing: border-box;

}


body {

  margin: 0;

}


/* Style the header */

.header {

  background-color: #f1f1f1;

  padding: 20px;

  text-align: center;

}


/* Style the top navigation bar */

.topnav {
  overflow: hidden;
  background-color: #333;
}

/* Style the topnav links */
.topnav a {
  float: left;
  display: block;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

/* Change color on hover */
.topnav a:hover {
  background-color: #ddd;
  color: black;
}

/* Create three unequal columns that floats next to each other */
.column {
  float: left;
  padding: 10px;
}
```

```
/* Left and right column */

.column.side {

  width: 25%;

}


/* Middle column */

.column.middle {

  width: 50%;

}


/* Clear floats after the columns */

.row::after {

  content: "";

  display: table;

  clear: both;

}


/* Responsive layout - makes the three columns stack on top of each
other instead of next to each other */

@media screen and (max-width: 600px) {

  .column.side, .column.middle {

    width: 100%;

  }

}

/* Style the footer */

.footer {

  background-color: #f1f1f1;
  padding: 10px;

  text-align: center;

}

</style>

</head>

<body>


<div class="header">

  <h1>Header</h1>

  <p>Resize the browser window to see the responsive effect.</p>

</div>


<div class="topnav">

  <a href="#">Link</a>

  <a href="#">Link</a>

  <a href="#">Link</a>

</div>


<div class="row">

  <div class="column side">

    <h2>Side</h2>

    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
elit..</p>

  </div>


  <div class="column middle">

    <h2>Main Content</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque
ultricies, eget elementum magna tristique. Quisque vehicula, risus
eget aliquam placerat, purus leo tincidunt eros, eget luctus quam
orci in velit. Praesent scelerisque tortor sed accumsan
convallis.</p>

    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque
ultricies, eget elementum magna tristique. Quisque vehicula, risus
eget aliquam placerat, purus leo tincidunt eros, eget luctus quam
orci in velit. Praesent scelerisque tortor sed accumsan
convallis.</p>

  </div>



  <div class="column side">

    <h2>Side</h2>

    <p>Lorem ipsum dolor sit amet, consectetur adipiscing
elit..</p>

  </div>

</div>



</body>

</html>
```

Footer

The footer is placed at the bottom of your page. It often contains information like copyright and contact info:

Example

```
<div class="footer">

  <p>Footer</p>

</div>
```

Responsive Website Layout

By using some of the CSS code above, we have created a responsive website layout, which varies between two columns and full-width columns depending on screen width:

```
<!DOCTYPE html>

<html>

<head>

<style>

* {

  box-sizing: border-box;

}

body {

  font-family: Arial;

  padding: 10px;

  background: #f1f1f1;

}

/* Header/Blog Title */

.header {

  padding: 30px;

  text-align: center;

  background: white;

}

.header h1 {

  font-size: 50px;

}

/* Style the top navigation bar */

.topnav {

  overflow: hidden;

  background-color: #333;
}

/* Style the topnav links */

.topnav a {

  float: left;

  display: block;

  color: #f2f2f2;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}

/* Change color on hover */

.topnav a:hover {

  background-color: #ddd;

  color: black;

}

/* Create two unequal columns that floats next to each other */

/* Left column */

.leftcolumn {

  float: left;

  width: 75%;

}


/* Right column */

.rightcolumn {

  float: left;

  width: 25%;

  background-color: #f1f1f1;

  padding-left: 20px;

}
```

```
/* Fake image */
.fakeimg {
  background-color: #aaa;
  width: 100%;
  padding: 20px;
}
/* Add a card effect for articles */
.card {
  background-color: white;
  padding: 20px;
  margin-top: 20px;
}

/* Clear floats after the columns */
.row::after {
  content: "";
  display: table;
  clear: both;
}
/* Footer */
.footer {
  padding: 20px;
  text-align: center;
  background: #ddd;
  margin-top: 20px;
}
```

```
/* Responsive layout - when the screen is less than 800px wide, make the two
columns stack on top of each other instead of next to each other */

@media screen and (max-width: 800px) {

  .leftcolumn, .rightcolumn {

    width: 100%;

    padding: 0;

  }

}

/* Responsive layout - when the screen is less than 400px wide, make the
navigation links stack on top of each other instead of next to each other */

@media screen and (max-width: 400px) {

  .topnav a {

    float: none;

    width: 100%;

  }

}

</style>

</head>

<body>



<div class="header">

  <h1>My Website</h1>

  <p>Resize the browser window to see the effect.</p>

</div>



<div class="topnav">

  <a href="#">Link</a>

  <a href="#">Link</a>

  <a href="#">Link</a>

  <a href="#" style="float:right">Link</a>
```

```
</div>



<div class="row">

  <div class="leftcolumn">

    <div class="card">

      <h2>TITLE HEADING</h2>

      <h5>Title description, Dec 7, 2017</h5>

      <div class="fakeimg" style="height:200px;">Image</div>

      <p>Some text..</p>

      <p>Sunt in culpa qui officia deserunt mollit anim id est laborum
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco.</p>

    </div>

    <div class="card">

      <h2>TITLE HEADING</h2>

      <h5>Title description, Sep 2, 2017</h5>

      <div class="fakeimg" style="height:200px;">Image</div>

      <p>Some text..</p>

      <p>Sunt in culpa qui officia deserunt mollit anim id est laborum
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco.</p>

    </div>

  </div>

  <div class="rightcolumn">

    <div class="card">

      <h2>About Me</h2>

      <div class="fakeimg" style="height:100px;">Image</div>

      <p>Some text about me in culpa qui officia deserunt mollit anim..</p>

    </div>

    <div class="card">
```

```
      <h3>Popular Post</h3>

      <div class="fakeimg"><p>Image</p></div>

      <div class="fakeimg"><p>Image</p></div>

      <div class="fakeimg"><p>Image</p></div>

    </div>

    <div class="card">

      <h3>Follow Me</h3>

      <p>Some text..</p>

    </div>

  </div>

</div>


<div class="footer">

  <h2>Footer</h2>

</div>


</body>

</html>
```

CSS Layout - The position Property

The `position` property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

The position Property

The `position` property specifies the type of positioning method used for an element.

There are five different position values:

- `static`
- `relative`
- `fixed`
- `absolute`
- `sticky`

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the `position` property is set first. They also work differently depending on the position value.

position: static;

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with `position: static;` is not positioned in any special way; it is always positioned according to the normal flow of the page:

This <div> element has position: static;

Here is the CSS that is used:

Example

```
<!DOCTYPE html>

<html>

<head>

<style>

div.static {

  position: static;

  border: 3px solid #73AD21;
}

</style>

</head>

<body>

<h2>position: static;</h2>

<p>An element with position: static; is not positioned in any
special way; it is always positioned according to the normal flow
of the page:</p>

<div class="static">

This div element has position: static;

</div>

</body>

</html>
```

position: relative;

An element with `position: relative;` is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

This <div> element has position: relative;

Here is the CSS that is used:

```
<!DOCTYPE html>

<html>

<head>

<style>

div.relative {

  position: relative;
  left: 30px;

  border: 3px solid #73AD21;

}

</style>

</head>

<body>

<h2>position: relative;</h2>

<p>An element with position: relative; is positioned relative to
its normal position:</p>

<div class="relative">

This div element has position: relative;

</div>

</body>

</html>
```

position: 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. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Notice the fixed element in the lower-right corner of the page. Here is the CSS that is used:

```
<!DOCTYPE html>

<html>

<head>

<style>

div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>
<h2>position: fixed;</h2>

<p>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:</p>
<div class="fixed">
This div element has position: fixed;
</div>
</body>
</html>
```

position: absolute;

An element with `position: absolute;` is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

Here is a simple example:

This <div> element has position: relative;
This <div> element has position: absolute;

Here is the CSS that is used:

Example

```
<!DOCTYPE html>

<html>

<head>

<style>

div.relative {

  position: relative;

  width: 400px;

  height: 200px;

  border: 3px solid #73AD21;

}


div.absolute {

  position: absolute;

  top: 80px;

  right: 0;

  width: 200px;

  height: 100px;

  border: 3px solid #73AD21;

}

</style>

</head>
```

```
<body>

<h2>position: absolute;</h2>

<p>An element with position: absolute; is positioned relative to
the nearest positioned ancestor (instead of positioned relative to
the viewport, like fixed):</p>

<div class="relative">This div element has position: relative;

  <div class="absolute">This div element has position:
absolute;</div>

</div>

</body>

</html>
```

position: sticky;

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Note: Internet Explorer does not support sticky positioning. Safari requires a -webkit- prefix (see example below). You must also specify at least one of `top`, `right`, `bottom` or `left` for sticky positioning to work.

In this example, the sticky element sticks to the top of the page (`top: 0`), when you reach its scroll position.

Example

```
<!DOCTYPE html>

<html>

<head>

<style>
```

```
div.sticky {

  position: -webkit-sticky;

  position: sticky;

  top: 0;

  padding: 5px;

  background-color: #cae8ca;

  border: 2px solid #4CAF50;

}

</style>

</head>

<body>

<p>Try to <b>scroll</b> inside this frame to understand how sticky
positioning works.</p>

<div class="sticky">I am sticky!</div>

<div style="padding-bottom:2000px">

  <p>In this example, the sticky element sticks to the top of the
page (top: 0), when you reach its scroll position.</p>

  <p>Scroll back up to remove the stickyness.</p>

  <p>Some text to enable scrolling.. Lorem ipsum dolor sit amet,
illum definitiones no quo, maluisset concludaturque et eum, altera
fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis
evertitur eum. Affert laboramus repudiandae nec et. Inciderint
efficiantur his ad. Eum no molestiae voluptatibus.</p>

  <p>Some text to enable scrolling.. Lorem ipsum dolor sit amet,
illum definitiones no quo, maluisset concludaturque et eum, altera
fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis
evertitur eum. Affert laboramus repudiandae nec et. Inciderint
efficiantur his ad. Eum no molestiae voluptatibus.</p>

</div>



</body>
```

```
</html>
```

Positioning Text In an Image

How to position text over an image:

Top-Left

```
<!DOCTYPE html>

<html>

<head>

<style>

.container {

  position: relative;

}


.topleft {

  position: absolute;

  top: 8px;

  left: 16px;

  font-size: 18px;

}


img {

  width: 100%;

  height: auto;

  opacity: 0.3;

}

</style>

</head>

<body>
```

```
<h2>Image Text</h2>

<p>Add some text to an image in the top left corner:</p>


<div class="container">

  <img src="img_5terre_wide.jpg" alt="Cinque Terre" width="1000" height="300">

  <div class="topleft">Top Left</div>

</div>


</body>

</html>
```

TOP-RIGHT

```
<!DOCTYPE html>

<html>

<head>

<style>

.container {

  position: relative;

}

.topright {

  position: absolute;

  top: 8px;

  right: 16px;

  font-size: 18px;

}

img {

  width: 100%;

  height: auto;

  opacity: 0.3;

}
```

```
</style>

</head>

<body>


<h2>Image Text</h2>

<p>Add some text to an image in the top right corner:</p>

<div class="container">

  <img src="img_5terre_wide.jpg" alt="Cinque Terre" width="1000" height="300">

  <div class="topright">Top Right</div>

</div>

</body>

</html>
```

BOTTOM-LEFT

```
<!DOCTYPE html>

<html>

<head>

<style>

.container {

  position: relative;

}

.bottomleft {

  position: absolute;

  bottom: 8px;

  left: 16px;

  font-size: 18px;

}

img {

  width: 100%;

  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>

<h2>Image Text</h2>
<p>Add some text to an image in the bottom left corner:</p>
<div class="container">
  <img src="img_5terre_wide.jpg" alt="Cinque Terre" width="1000" height="300">
  <div class="bottomleft">Bottom Left</div>
</div>
</body>
</html>
```

BOTTOM-RIGHT

```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.bottomright {
  position: absolute;
  bottom: 8px;
  right: 16px;
  font-size: 18px;
}
img {
  width: 100%;
  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>

<h2>Image Text</h2>
<p>Add some text to an image in the bottom right corner:</p>
<div class="container">
  <img src="img_5terre_wide.jpg" alt="Cinque Terre" width="1000" height="300">
  <div class="bottomright">Bottom Right</div>
</div>
</body>
</html>
```

CENTERED

```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.center {
  position: absolute;
  top: 50%;
  width: 100%;
  text-align: center;
  font-size: 18px;
}
img {
  width: 100%;
  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Center text in image:</p>
<div class="container">
  <img src="img_5terre_wide.jpg" alt="Cinque Terre" width="1000" height="300">
  <div class="center">Centered</div>
</div>
</body>
</html>
```

Exercise:

Position the <h1> element to always be 50px from the top, and 10px from the right, relative to the window/frame edges.

```
<style>
h1 {
  : ;
  : 50px;
  : 10px;
}
</style>
```

```
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph</p>
  <p>This is a paragraph</p>
</body>
```

All CSS Positioning Properties

Property	Description
bottom	Sets the bottom margin edge for a positioned box
clip	Clips an absolutely positioned element
left	Sets the left margin edge for a positioned box
position	Specifies the type of positioning for an element
right	Sets the right margin edge for a positioned box
top	Sets the top margin edge for a positioned box

Week 4:

HTML class Attribute

The HTML `class` attribute is used to specify a class for an HTML element.

Multiple HTML elements can share the same class.

Using The class Attribute

The `class` attribute is often used to point to a class name in a style sheet. It can also be used by a JavaScript to access and manipulate elements with the specific class name.

In the following example we have three `<div>` elements with a `class` attribute with the value of "city". All of the three `<div>` elements will be styled equally according to the .city style definition in the head section:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
.city {
  background-color: tomato;
  color: white;
  border: 2px solid black;
  margin: 20px;
  padding: 20px;
}
</style>
</head>
<body>

<div class="city">
  <h2>London</h2>
  <p>London is the capital of England.</p>
</div>

<div class="city">
  <h2>Paris</h2>
  <p>Paris is the capital of France.</p>
</div>

<div class="city">
  <h2>Tokyo</h2>
  <p>Tokyo is the capital of Japan.</p>
</div>

</body>
</html>
```

In the following example we have two `<span>` elements with a `class` attribute with the value of "note". Both `<span>` elements will be styled equally according to the `.note` style definition in the head section:

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

```
<style>
.note {
  font-size: 120%;
  color: red;
}
</style>
</head>
<body>

<h1>My <span class="note">Important</span> Heading</h1>
<p>This is some <span class="note">important</span> text.</p>

</body>
</html>
```

Tip: The `class` attribute can be used on **any** HTML element.

Note: The class name is case sensitive!

The Syntax For Class

To create a class; write a period (.) character, followed by a class name. Then, define the CSS properties within curly braces {}:

Example

Create a class named "city":

```
<!DOCTYPE html>
<html>
<head>
<style>
.city {
  background-color: tomato;
  color: white;
  padding: 10px;
}
</style>
</head>
<body>

<h2 class="city">London</h2>
<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>
<p>Paris is the capital of France.</p>
```

```
<h2 class="city">Tokyo</h2>
<p>Tokyo is the capital of Japan.</p>

</body>
</html>
```

Multiple Classes

HTML elements can belong to more than one class.

To define multiple classes, separate the class names with a space, e.g. <div class="city main">. The element will be styled according to all the classes specified.

In the following example, the first `<h2>` element belongs to both the `city` class and also to the `main` class, and will get the CSS styles from both of the classes:

```
<h2 class="city main">London</h2>
<h2 class="city">Paris</h2>
<h2 class="city">Tokyo</h2>
```

Different Elements Can Share Same Class

Different HTML elements can point to the same class name.

In the following example, both `<h2>` and `<p>` point to the "city" class and will share the same style:

Example

```
<h2 class="city">Paris</h2>
<p class="city">Paris is the capital of France</p>
```

Use of The class Attribute in JavaScript

The class name can also be used by JavaScript to perform certain tasks for specific elements.

JavaScript can access elements with a specific class name with the `getElementsByClassName()` method:

Example

Click on a button to hide all elements with the class name "city":

```
<!DOCTYPE html>

<html>

<body>

<h2>Use of The class Attribute in JavaScript</h2>

<p>Click the button to hide all elements with class name
"city":</p>



<button onclick="myFunction()">Hide elements</button>

<h2 class="city">London</h2>

<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>

<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>

<p>Tokyo is the capital of Japan.</p>

<script>

function myFunction() {

  var x = document.getElementsByClassName("city");

  for (var i = 0; i < x.length; i++) {

    x[i].style.display = "none";

  }

}

</script>

</body>
```

```
</html>
```

Don't worry if you don't understand the code in the example above.

You will learn more about JavaScript in coming lessons

Chapter Summary

- The HTML `class` attribute specifies one or more class names for an element
- Classes are used by CSS and JavaScript to select and access specific elements
- The `class` attribute can be used on any HTML element
- The class name is case sensitive
- Different HTML elements can point to the same class name
- JavaScript can access elements with a specific class name with the `getElementsByClassName()` method

Exercise:

Create a class selector named "special".

Add a color property with the value "blue" inside the "special" class.

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

  ;

</style>
</head>
<body>

<p class="special">My paragraph</p>

</body>
</html>
```

HTML id Attribute

The HTML id attribute is used to specify a unique id for an HTML element.

You cannot have more than one element with the same id in an HTML document.

Using The id Attribute

The id attribute specifies a unique id for an HTML element. The value of the id attribute must be unique within the HTML document.

The id attribute is used to point to a specific style declaration in a style sheet. It is also used by JavaScript to access and manipulate the element with the specific id.

The syntax for id is: write a hash character (#), followed by an id name. Then, define the CSS properties within curly braces {}.

In the following example we have an `<h1>` element that points to the id name "myHeader". This `<h1>` element will be styled according to the `#myHeader` style definition in the head section:

```
<!DOCTYPE html>
<html>
<head>
<style>
#myHeader {
  background-color: lightblue;
  color: black;
  padding: 40px;
  text-align: center;
}
</style>
</head>
<body>

<h1 id="myHeader">My Header</h1>

</body>
</html>
```

Note: The id name is case sensitive!

Note: The id name must contain at least one character, cannot start with a number, and must not contain whitespaces (spaces, tabs, etc.).

Difference Between Class and ID

A class name can be used by multiple HTML elements, while an id name must only be used by one HTML element within the page:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
/* Style the element with the id "myHeader" */
#myHeader {
  background-color: lightblue;
  color: black;
  padding: 40px;
  text-align: center;
}

/* Style all elements with the class name "city" */
.city {
  background-color: tomato;
  color: white;
  padding: 10px;
}
</style>
</head>
<body>

<h2>Difference Between Class and ID</h2>
<p>A class name can be used by multiple HTML elements, while an id name must only be used by one HTML element
within the page:</p>

<!-- An element with a unique id -->
<h1 id="myHeader">My Cities</h1>

<!-- Multiple elements with same class -->
<h2 class="city">London</h2>
<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>
<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>
<p>Tokyo is the capital of Japan.</p>

</body>
```

```
</html>
```

Using The id Attribute in JavaScript

The `id` attribute can also be used by JavaScript to perform some tasks for that specific element.

JavaScript can access an element with a specific id with the `getElementById()` method:

Example

Use the `id` attribute to manipulate text with JavaScript:

```
<!DOCTYPE html>
<html>
<body>
<h2>Using The id Attribute in JavaScript</h2>
<p>JavaScript can access an element with a specified id by using
the getElementById() method:</p>

<h1 id="myHeader">Hello World!</h1>
<button onclick="displayResult()">Change text</button>

<script>
function displayResult() {
  document.getElementById("myHeader").innerHTML = "Have a nice
day!";
}
</script>
</body>
</html>
```

Chapter Summary

- The id attribute is used to specify a unique id for an HTML element
- The value of the id attribute must be unique within the HTML document
- The id attribute is used by CSS and JavaScript to style/select a specific element
- The value of the id attribute is case sensitive
- The id attribute is also used to create HTML bookmarks
- JavaScript can access an element with a specific id with the getElementById() method

Exercise:

Add the correct HTML attribute to make the H1 element red.

```
<!DOCTYPE html>
<html>
<head>
<style>
#myheader {color:red;}
</style>
</head>
<body>

<h1 >My Home Page</h1>

</body>
</html>
```

HTML Semantic Elements

Semantic elements = elements with a meaning.

What are Semantic Elements?

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of **non-semantic** elements: `<div>` and `<span>` - Tells nothing about its content.

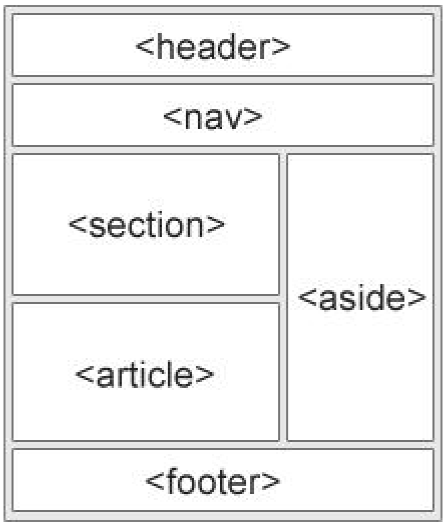
Examples of **semantic** elements: `<form>`, `<table>`, and `<article>` - Clearly defines its content.

Semantic Elements in HTML

Many web sites contain HTML code like: <div id="nav"> <div class="header"> <div id="footer"> to indicate navigation, header, and footer.

In HTML there are some semantic elements that can be used to define different parts of a web page:

- <article>
- <aside>
- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>



HTML <section> Element

The `<section>` element defines a section in a document.

According to W3C's HTML documentation: "A section is a thematic grouping of content, typically with a heading."

Examples of where a `<section>` element can be used:

- Chapters
- Introduction
- News items
- Contact information

A web page could normally be split into sections for introduction, content, and contact information.

Example

Two sections in a document:

```
<!DOCTYPE html>

<html>

<body>

<section>

  <h1>WWF</h1>

  <p>The World Wide Fund for Nature (WWF) is an international organization working on issues regarding the conservation, research and restoration of the environment, formerly named the World Wildlife Fund. WWF was founded in 1961.</p>

</section>

<section>

  <h1>WWF's Panda symbol</h1>

  <p>The Panda has become the symbol of WWF. The well-known panda logo of WWF originated from a panda named Chi Chi that was transferred from the Beijing Zoo to the London Zoo in the same year of the establishment of WWF.</p>

</section>

</body>
```

```
</html>
```

HTML <article> Element

The `<article>` element specifies independent, self-contained content.

An article should make sense on its own, and it should be possible to distribute it independently from the rest of the web site.

Examples of where the `<article>` element can be used:

- Forum posts
- Blog posts
- User comments
- Product cards
- Newspaper articles

Example

Three articles with independent, self-contained content:

```
<!DOCTYPE html>

<html>
<head>
<style>
.all-browsers {
  margin: 0;
  padding: 5px;
  background-color: lightgray;
}

.all-browsers > h1, .browser {
  margin: 10px;
  padding: 5px;
}

.browser {
  background: white;
}

.browser > h2, p {
  margin: 4px;
  font-size: 90%;
}
</style>
</head>
<body>

<article class="all-browsers">
  <h1>Most Popular Browsers</h1>
  <article class="browser">
    <h2>Google Chrome</h2>
    <p>Google Chrome is a web browser developed by Google, released
in 2008. Chrome is the world's most popular web browser today!</p>
  </article>
  <article class="browser">
    <h2>Mozilla Firefox</h2>
    <p>Mozilla Firefox is an open-source web browser developed by
Mozilla. Firefox has been the second most popular web browser since
January, 2018.</p>
  </article>
  <article class="browser">
    <h2>Microsoft Edge</h2>
    <p>Microsoft Edge is a web browser developed by Microsoft,
released in 2015. Microsoft Edge replaced Internet Explorer.</p>
  </article>
</article>

</body>
</html>
```

Nesting <article> in <section> or Vice Versa?

The `<article>` element specifies independent, self-contained content.

The `<section>` element defines section in a document.

Can we use the definitions to decide how to nest those elements? No, we cannot!

So, you will find HTML pages with `<section>` elements containing `<article>` elements, and `<article>` elements containing `<section>` elements.

HTML <header> Element

The `<header>` element represents a container for introductory content or a set of navigational links.

A `<header>` element typically contains:

- one or more heading elements (<h1> - <h6>)
- logo or icon
- authorship information

Note: You can have several `<header>` elements in one HTML document. However, `<header>` cannot be placed within a `<footer>`, `<address>` or another `<header>` element.

Example

A header for an <article>:

```
<article>
  <header>
    <h1>What Does WWF Do?</h1>
    <p>WWF's mission:</p>
  </header>
  <p>WWF's mission is to stop the degradation of our planet's natural
environment,
  and build a future in which humans live in harmony with nature.</p>
</article>
```

HTML <footer> Element

The `<footer>` element defines a footer for a document or section.

A `<footer>` element typically contains:

- authorship information
- copyright information
- contact information
- sitemap
- back to top links
- related documents

You can have several `<footer>` elements in one document.

Example

A footer section in a document:

```
<footer>
  <p>Author: Hege Refsnes</p>
  <p><a href="mailto:hege@example.com">hege@example.com</a></p>
</footer>
```

HTML <nav> Element

The `<nav>` element defines a set of navigation links.

Notice that NOT all links of a document should be inside a `<nav>` element. The `<nav>` element is intended only for major blocks of navigation links.

Browsers, such as screen readers for disabled users, can use this element to determine whether to omit the initial rendering of this content.

Example

A set of navigation links:

```
<nav>
  <a href="/html/">HTML</a> |
  <a href="/css/">CSS</a> |
  <a href="/js/">JavaScript</a> |
  <a href="/jquery/">jQuery</a>
</nav>
```

HTML <aside> Element

The `<aside>` element defines some content aside from the content it is placed in (like a sidebar).

The <aside> content should be indirectly related to the surrounding content.

Example

Display some content aside from the content it is placed in with css styling:

```
<html>
<head>
<style>
aside {
  width: 30%;
  padding-left: 15px;
  margin-left: 15px;
  float: right;
  font-style: italic;
  background-color: lightgray;
}
</style>
</head>
<body>

<p>My family and I visited The Epcot center this summer. The
weather was nice, and Epcot was amazing! I had a great summer
together with my family!</p>

<aside>
<p>The Epcot center is a theme park at Walt Disney World Resort
featuring exciting attractions, international pavilions, award-
winning fireworks and seasonal special events.</p>
</aside>

<p>My family and I visited The Epcot center this summer. The
weather was nice, and Epcot was amazing! I had a great summer
together with my family!</p>
<p>My family and I visited The Epcot center this summer. The
weather was nice, and Epcot was amazing! I had a great summer
together with my family!</p>

</body>
</html>
```

HTML <figure> and <figcaption> Elements

The `<figure>` tag specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.

The `<figcaption>` tag defines a caption for a `<figure>` element. The `<figcaption>` element can be placed as the first or as the last child of a `<figure>` element.

The `<img>` element defines the actual image/illustration.

```
<figure>
  <img src="pic_trulli.jpg" alt="Trulli">
  <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption>
</figure>
```

Why Semantic Elements?

According to the W3C: "A semantic Web allows data to be shared and reused across applications, enterprises, and communities."

Semantic Elements in HTML

Below is a list of some of the semantic elements in HTML.

Tag	Description
<article>	Defines independent, self-contained content
<aside>	Defines content aside from the page content
<details>	Defines additional details that the user can view or hide
<figcaption>	Defines a caption for a <figure> element
<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for a document or section

<header>	Specifies a header for a document or section
<main>	Specifies the main content of a document
<mark>	Defines marked/highlighted text
<nav>	Defines navigation links
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time

HTML Layout Elements and Techniques

Websites often display content in multiple columns (like a magazine or a newspaper).

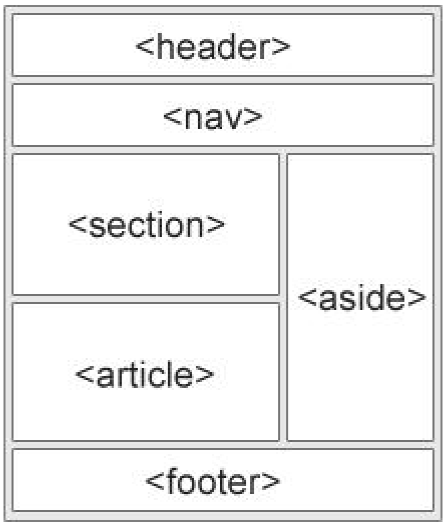
```
<!DOCTYPE html>

<html lang="en">

<head>

<title>CSS Template</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<style>

* {

  box-sizing: border-box;

}

body {

  font-family: Arial, Helvetica, sans-serif;
}


/* Style the header */
header {
  background-color: #666;
  padding: 30px;
  text-align: center;
  font-size: 35px;
  color: white;
}
/* Create two columns/boxes that floats next to each other */
nav {
  float: left;
  width: 30%;
  height: 300px; /* only for demonstration, should be removed */
  background: #ccc;
  padding: 20px;
}
/* Style the list inside the menu */
nav ul {
  list-style-type: none;
  padding: 0;
}
article {
  float: left;
  padding: 20px;
  width: 70%;
  background-color: #f1f1f1;
  height: 300px; /* only for demonstration, should be removed */
}
```

```
/* Clear floats after the columns */

section::after {

  content: "";

  display: table;

  clear: both;

}

/* Style the footer */

footer {

  background-color: #777;

  padding: 10px;

  text-align: center;

  color: white;

}

/* Responsive layout - makes the two columns/boxes stack on top of each other instead of
next to each other, on small screens */

@media (max-width: 600px) {

  nav, article {

    width: 100%;

    height: auto;

  }

}

</style>

</head>

<body>

<h2>CSS Layout Float</h2>

<p>In this example, we have created a header, two columns/boxes and a footer. On smaller
screens, the columns will stack on top of each other.</p>

<p>Resize the browser window to see the responsive effect (you will learn more about this in
our next chapter - HTML Responsive.)</p>

<header>

  <h2>Cities</h2>
```

```
</header>

<section>

  <nav>

    <ul>

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

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

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

    </ul>

  </nav>

  <article>

    <h1>London</h1>

    <p>London is the capital city of England. It is the most populous city in the  United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>

    <p>Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.</p>

  </article>

</section>

<footer>

  <p>Footer</p>

</footer>

</body>

</html>
```

HTML Layout Elements

HTML has several semantic elements that define the different parts of a web page:



- `<header>` - Defines a header for a document or a section
- `<nav>` - Defines a set of navigation links
- `<section>` - Defines a section in a document
- `<article>` - Defines an independent, self-contained content
- `<aside>` - Defines content aside from the content (like a sidebar)
- `<footer>` - Defines a footer for a document or a section
- `<details>` - Defines additional details that the user can open and close on demand
- `<summary>` - Defines a heading for the `<details>` element

HTML Layout Techniques

There are four different techniques to create multicolumn layouts. Each technique has its pros and cons:

- CSS framework
- CSS float property
- CSS flexbox
- CSS grid

CSS Frameworks

If you want to create your layout fast, you can use a CSS framework, like W3.CSS or Bootstrap or Tailwind CSS

CSS Float Layout

It is common to do entire web layouts using the CSS `float` property. Float is easy to learn - you just need to remember how the `float` and `clear` properties work. **Disadvantages:** Floating elements are tied to the document flow, which may harm the flexibility.

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
<title>CSS Template</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
* {
  box-sizing: border-box;
}

body {
  font-family: Arial, Helvetica, sans-serif;
}

/* Style the header */
header {
  background-color: #666;
  padding: 30px;
  text-align: center;
  font-size: 35px;
  color: white;
}

/* Create two columns/boxes that floats next to each other */
nav {
  float: left;
  width: 30%;
  height: 300px; /* only for demonstration, should be removed */
  background: #ccc;
  padding: 20px;
}
```

```
/* Style the list inside the menu */

nav ul {

  list-style-type: none;

  padding: 0;

}


article {

  float: left;

  padding: 20px;

  width: 70%;

  background-color: #f1f1f1;

  height: 300px; /* only for demonstration, should be removed */

}


/* Clear floats after the columns */

section::after {

  content: "";

  display: table;

  clear: both;

}


/* Style the footer */

footer {

  background-color: #777;

  padding: 10px;

  text-align: center;

  color: white;

}
```

```
/* Responsive layout - makes the two columns/boxes stack on top of each other instead of
next to each other, on small screens */

@media (max-width: 600px) {

  nav, article {

    width: 100%;

    height: auto;

  }

}

</style>

</head>

<body>



<h2>CSS Layout Float</h2>

<p>In this example, we have created a header, two columns/boxes and a footer. On smaller
screens, the columns will stack on top of each other.</p>

<p>Resize the browser window to see the responsive effect (you will learn more about this in
our next chapter - HTML Responsive.)</p>

<header>

  <h2>Cities</h2>

</header>





<section>

  <nav>

    <ul>

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

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

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

    </ul>

  </nav>

  <article>
```

```
    <h1>London</h1>

    <p>London is the capital city of England. It is the most populous city in the  United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>

    <p>Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.</p>

  </article>

</section>

<footer>

  <p>Footer</p>

</footer>

</body>

</html>
```

CSS Flexbox Layout

Use of flexbox ensures that elements behave predictably when the page layout must accommodate different screen sizes and different display devices.

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>CSS Template</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<style>

* {

  box-sizing: border-box;

}


body {

  font-family: Arial, Helvetica, sans-serif;

}
```

```
/* Style the header */
header {
  background-color: #666;
  padding: 30px;
  text-align: center;
  font-size: 35px;
  color: white;
}

/* Container for flexboxes */
section {
  display: -webkit-flex;
  display: flex;
}

/* Style the navigation menu */
nav {
  -webkit-flex: 1;
  -ms-flex: 1;
  flex: 1;
  background: #ccc;
  padding: 20px;
}

/* Style the list inside the menu */
nav ul {
  list-style-type: none;
  padding: 0;
}
```

```
/* Style the content */

article {

  -webkit-flex: 3;

  -ms-flex: 3;

  flex: 3;

  background-color: #f1f1f1;

  padding: 10px;

}


/* Style the footer */

footer {

  background-color: #777;

  padding: 10px;

  text-align: center;

  color: white;

}


/* Responsive layout - makes the menu and the content (inside the section) sit on top of
each other instead of next to each other */

@media (max-width: 600px) {

  section {

    -webkit-flex-direction: column;

    flex-direction: column;

  }

}

</style>

</head>

<body>
```

```
<h2>CSS Layout Flexbox</h2>

<p>In this example, we have created a header, two columns/boxes and a footer. On smaller screens, the columns will stack on top of each other.</p>

<p>Resize the browser window to see the responsive effect.</p>



<header>

  <h2>Cities</h2>

</header>



<section>

  <nav>

    <ul>

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

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

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

    </ul>

  </nav>



  <article>

    <h1>London</h1>

    <p>London is the capital city of England. It is the most populous city in the  United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>

    <p>Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.</p>

  </article>

</section>



<footer>

  <p>Footer</p>

</footer>
```

```
</body>

</html>
```

CSS Grid Layout

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

HTML Responsive Web Design

Responsive web design is about creating web pages that look good on all devices!

A responsive web design will automatically adjust for different screen sizes and viewports.

What is Responsive Web Design?

Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones):

Setting The Viewport

To create a responsive website, add the following `<meta>` tag to all your web pages:

Example

```
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

Responsive Images

Responsive images are images that scale nicely to fit any browser size.

Using the width Property

If the CSS `width` property is set to 100%, the image will be responsive and scale up and down:

```
<img src="img_girl.jpg" style="width:100%;">
```

Notice that in the example above, the image can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the `max-width` property instead.

Using the max-width Property

If the `max-width` property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

```
<img src="img_girl.jpg" style="max-width:100%;height:auto;">
```

Show Different Images Depending on Browser Width

The HTML `<picture>` element allows you to define different images for different browser window sizes.

Resize the browser window to see how the image below changes depending on the width:

```
<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-
scale=1.0">

</head>

<body>
```

```
<h2>Show Different Images Depending on Browser Width</h2>

<p>Resize the browser width and the image will change at 600px and 
1500px.</p>


<picture>

  <source srcset="img_smallflower.jpg" media="(max-width: 600px)">

  <source srcset="img_flowers.jpg" media="(max-width: 1500px)">

  <source srcset="flowers.jpg">

  <img src="img_flowers.jpg" alt="Flowers" style="width:auto;">

</picture>


</body>

</html>
```

Responsive Text Size

The text size can be set with a "vw" unit, which means the "viewport width".

That way the text size will follow the size of the browser window:

```
<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-
scale=1.0">

</head>

<body>

<h1 style="font-size:10vw;">Responsive Text</h1>

<p style="font-size:5vw;">Resize the browser window to see how the 
text size scales.</p>
```

```
<p style="font-size:5vw;">Use the "vw" unit when sizing the text.
10vw will set the size to 10% of the viewport width.</p>

</body>

</html>
```

Viewport is the browser window size. 1vw = 1% of viewport width. If the viewport is 50cm wide, 1vw is 0.5cm.

Media Queries

In addition to resize text and images, it is also common to use media queries in responsive web pages.

With media queries you can define completely different styles for different browser sizes.

Example: resize the browser window to see that the three div elements below will display horizontally on large screens and stack vertically on small screens:

```
<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-
scale=1.0">

<style>

* {

  box-sizing: border-box;

}


.left {

  background-color: #2196F3;

  padding: 20px;

  float: left;
  width: 20%; /* The width is 20%, by default */
}

.main {
  background-color: #f1f1f1;
  padding: 20px;
  float: left;
  width: 60%; /* The width is 60%, by default */
}

.right {
  background-color: #04AA6D;
  padding: 20px;
  float: left;
  width: 20%; /* The width is 20%, by default */
}

/* Use a media query to add a break point at 800px: */
@media screen and (max-width: 800px) {
  .left, .main, .right {
    width: 100%; /* The width is 100%, when the viewport is 800px
or smaller */
  }
}
</style>
</head>
<body>
```

```
<h2>Media Queries</h2>

<p>Resize the browser window.</p>

<p>Make sure you reach the breakpoint at 800px when resizing this
frame.</p>

<div class="left">

  <p>Left Menu</p>

</div>

<div class="main">

  <p>Main Content</p>

</div>

<div class="right">

  <p>Right Content</p>

</div>

</body>

</html>
```

Responsive Web Page - Full Example

A responsive web page should look good on large desktop screens and on small mobile phones.

```
<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<style>

* {

  box-sizing: border-box;

}

.menu {
  float: left;

  width: 20%;

  text-align: center;

}


.menu a {

  background-color: #e5e5e5;

  padding: 8px;

  margin-top: 7px;

  display: block;

  width: 100%;

  color: black;

}

.main {

  float: left;

  width: 60%;

  padding: 0 20px;

}

.right {

  background-color: #e5e5e5;

  float: left;

  width: 20%;

  padding: 15px;

  margin-top: 7px;

  text-align: center;

}

@media only screen and (max-width: 620px) {

  /* For mobile phones: */

  .menu, .main, .right {

    width: 100%;
  }

}

</style>

</head>

<body style="font-family:Verdana;color:#aaaaaa;">



<div style="background-color:#e5e5e5;padding:15px;text-align:center;">

  <h1>Hello World</h1>

</div>

<div style="overflow:auto">

  <div class="menu">

    <a href="#">Link 1</a>

    <a href="#">Link 2</a>

    <a href="#">Link 3</a>

    <a href="#">Link 4</a>

  </div>

  <div class="main">

    <h2>Lorum Ipsum</h2>

    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>

  </div>

  <div class="right">

    <h2>About</h2>

    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>

  </div>

</div>

<div style="background-color:#e5e5e5;text-align:center;padding:10px;margin-top:7px;">©
copyright w3schools.com</div>

</body>

</html>
```

CSS Flexbox

CSS3 Flexbox

Flexible boxes, or flexbox, is a new layout mode in CSS3.

Use of flexbox ensures that elements behave predictably when the page layout must accommodate different screen sizes and different display devices.

For many applications, the flexible box model provides an improvement over the block model in that it does not use floats, nor do the flex container's margins collapse with the margins of its contents.

CSS3 Flexbox Concepts

Flexbox consists of flex containers and flex items.

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 is one or more flex items.

Note: Everything outside a flex container and inside a flex item is rendered as usual. Flexbox defines how flex items are laid out inside a flex container.

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:

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
    display: -webkit-flex;
    display: flex;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}

.flex-item {
    background-color: cornflowerblue;
    width: 100px;
    height: 100px;
    margin: 10px;
}
</style>
</head>
<body>

<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>

</body>
</html>
```

CSS Flex Container

The flex container properties are:

- `flex-direction`
- `flex-wrap`
- `flex-flow`
- `justify-content`
- `align-items`
- `align-content`

It is also possible to change the direction of the flex line.

If we set the *direction* property to *rtl* (right-to-left), the text is drawn right to left, and also the flex line changes direction, which will change the page layout:

```
body {
    direction: rtl;
}

.flex-container {
    display: -webkit-flex;
    display: flex;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}

.flex-item {
    background-color: cornflowerblue;
    width: 100px;
    height: 100px;
    margin: 10px;
}
```

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, top-to-bottom).

The other values are as follows:

- `row-reverse` - If the writing-mode (direction) is left to right, the flex items will be laid out right to left
- `column` - If the writing system is horizontal, the flex items will be laid out vertically
- `column-reverse` - Same as column, but reversed

The following example shows the result of using the `row-reverse` value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-direction: row-reverse;
    flex-direction: row-reverse;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the `column` value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-direction: column;
    flex-direction: column;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the `column-reverse` value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-direction: column-reverse;
    flex-direction: column-reverse;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The justify-content Property

The `justify-content` property horizontally aligns the flexible container's items when the items do not use all available space on the main-axis.

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

The following example shows the result of using the `flex-end` value:

Example

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-justify-content: flex-end;
    justify-content: flex-end;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the `center` value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-justify-content: center;
    justify-content: center;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the `space-between` value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-justify-content: space-between;
    justify-content: space-between;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the *space-around* value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-justify-content: space-around;
    justify-content: space-around;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The align-items Property

The *align-items* property vertically aligns the flexible container's items when the items do not use all available space on the cross-axis.

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 positioned at the baseline of the container

The following example shows the result of using the *stretch* value (this is the default value):

Example

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-align-items: stretch;
    align-items: stretch;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the *flex-start* value:

Example

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-align-items: flex-start;
    align-items: flex-start;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the *flex-end* value:

Example

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-align-items: flex-end;
    align-items: flex-end;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the *center* value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-align-items: center;
    align-items: center;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the *baseline* value:

Example

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-align-items: baseline;
    align-items: baseline;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
}
```

The flex-wrap Property

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

The following example shows the result of using the *nowrap* value (this is the default value):

Example

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-wrap: nowrap;
    flex-wrap: nowrap;
    width: 300px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the `wrap` value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-wrap: wrap;
    flex-wrap: wrap;
    width: 300px;
    height: 250px;
    background-color: lightgrey;
}
```

The following example shows the result of using the `wrap-reverse` value:

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-wrap: wrap-reverse;
    flex-wrap: wrap-reverse;
    width: 300px;
    height: 250px;
    background-color: lightgrey;
}
```

The align-content Property

The `align-content` property modifies the behavior of the `flex-wrap` property. It is similar to `align-items`, but instead of aligning flex items, it aligns flex lines.

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

The following example shows the result of using the `center` value:

Example

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-wrap: wrap;
    flex-wrap: wrap;
    -webkit-align-content: center;
    align-content: center;
    width: 300px;
    height: 300px;
    background-color: lightgrey;
}
```

Flex Item Properties

Ordering

The `order` property specifies the order of a flexible item relative to the rest of the flexible items inside the same container:

Example

```
.flex-item {
    background-color: cornflowerblue;
    width: 100px;
    height: 100px;
    margin: 10px;
}

.first {
    -webkit-order: -1;
    order: -1;
}
```

Margin

Setting *margin: auto;* will absorb extra space. It can be used to push flex items into different positions.

In the following example we set *margin-right: auto;* on the first flex item. This will cause all the extra space to be absorbed to the right of that element:

Example

```
.flex-item {
    background-color: cornflowerblue;
    width: 75px;
    height: 75px;
    margin: 10px;
}

.flex-item:first-child {
    margin-right: auto;
}
```

Perfect Centering

In the following example we will solve an almost daily problem: perfect centering.

It is very easy with flexbox. Setting *margin: auto;* will make the item perfectly centered in both axis:

Example

```
.flex-item {
    background-color: cornflowerblue;
    width: 75px;
    height: 75px;
    margin: auto;
}
```

align-self

The `align-self` property of flex items overrides the flex container's align-items property for that item. It has the same possible values as the `align-items` property.

The following example sets different align-self values to each flex item:

Example

```
.flex-item {
    background-color: cornflowerblue;
    width: 60px;
    min-height: 100px;
    margin: 10px;
}

.item1 {
    -webkit-align-self: flex-start;
    align-self: flex-start;
}
.item2 {
    -webkit-align-self: flex-end;
    align-self: flex-end;
}

.item3 {
    -webkit-align-self: center;
    align-self: center;
}

.item4 {
    -webkit-align-self: baseline;
    align-self: baseline;
}

.item5 {
    -webkit-align-self: stretch;
    align-self: stretch;
}
```

flex

The `flex` property specifies the length of the flex item, relative to the rest of the flex items inside the same container.

In the following example, the first flex item will consume 2/4 of the free space, and the other two flex items will consume 1/4 of the free space each:

Example

```
.flex-item {
    background-color: cornflowerblue;
    margin: 10px;
}

.item1 {
    -webkit-flex: 2;
    flex: 2;
}

.item2 {
    -webkit-flex: 1;
    flex: 1;
}

.item3 {
    -webkit-flex: 1;
    flex: 1;
}
```

CSS Flex Items

Child Elements (Items)

The direct child elements of a flex container automatically becomes flexible (flex) items.

```
<!DOCTYPE html>

<html>

<head>

<style>

.flex-container {

  display: flex;

  background-color: #f1f1f1;

}

.flex-container > div {

  background-color: DodgerBlue;

  color: white;

  width: 100px;

  margin: 10px;

  text-align: center;

  line-height: 75px;

  font-size: 30px;

}

</style>

</head>

<body>

<h1>Flexible Items</h1>
```

```
<div class="flex-container">

  <div>1</div>

  <div>2</div>

  <div>3</div>

  <div>4</div>

</div>

<p>All direct children of a flexible container becomes flexible items.</p>

</body>

</html>
```

The flex item properties are:

- order
- flex-grow
- flex-shrink
- flex-basis
- flex
- align-self

The order Property

The `order` property specifies the order of the flex items.

The first flex item in the code does not have to appear as the first item in the layout.

The order value must be a number, default value is 0.

Example

The *order* property can change the order of the flex items:

```
<div class="flex-container">
  <div style="order: 3">1</div>
  <div style="order: 2">2</div>
  <div style="order: 4">3</div>
  <div style="order: 1">4</div>
</div>
```

The flex-grow Property

The flex-grow property specifies how much a flex item will grow relative to the rest of the flex items.

The value must be a number, default value is 0.

Example

Make the third flex item grow eight times faster than the other flex items:

```
<div class="flex-container">
  <div style="flex-grow: 1">1</div>
  <div style="flex-grow: 1">2</div>
  <div style="flex-grow: 8">3</div>
</div>
```

The flex-shrink Property

The flex-shrink property specifies how much a flex item will shrink relative to the rest of the flex items.

The value must be a number, default value is 1.

Example

Do not let the third flex item shrink as much as the other flex items:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-shrink: 0">3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
  <div>10</div>
</div>
```

The flex-basis Property

The flex-basis property specifies the initial length of a flex item.

Example

Set the initial length of the third flex item to 200 pixels:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-basis: 200px">3</div>
  <div>4</div>
</div>
```

The flex Property

The `flex` property is a shorthand property for the `flex-grow`, `flex-shrink`, and `flex-basis` properties.

Example

Make the third flex item not growable (0), not shrinkable (0), and with an initial length of 200 pixels:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex: 0 0 200px">3</div>
  <div>4</div>
</div>
```

The align-self Property

The `align-self` property specifies the alignment for the selected item inside the flexible container.

The `align-self` property overrides the default alignment set by the container's `align-items` property.

In these examples we use a 200 pixels high container, to better demonstrate the `align-self` property:

Example

Align the third flex item in the middle of the container:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="align-self: center">3</div>
  <div>4</div>
</div>
```

Example

Align the second flex item at the top of the container, and the third flex item at the bottom of the container:

```
<div class="flex-container">
  <div>1</div>
  <div style="align-self: flex-start">2</div>
  <div style="align-self: flex-end">3</div>
  <div>4</div>
</div>
```

CSS3 Flexbox Properties

The following table lists the CSS properties used with flexbox:

Property	Description
display	Specifies the type of box used for an HTML element
flex-direction	Specifies the direction of the flexible items inside a flex container
justify-content	Horizontally aligns the flex items when the items do not use all availab the main-axis
align-items	Vertically aligns the flex items when the items do not use all available s cross-axis
flex-wrap	Specifies whether the flex items should wrap or not, if there is not enou on one flex line
align-content	Modifies the behavior of the flex-wrap property. It is similar to align-ite of aligning flex items, it aligns flex lines
flex-flow	A shorthand propert for flex-direction and flex-wrap
order	Specifies the order of a flexible item relative to the rest of the flex item container

align-self	Used on flex items. Overrides the container's align-items property
flex	Specifies the length of a flex item, relative to the rest of the flex items container

Property	Description
align-self	Specifies the alignment for a flex item (overrides the flex container's al
flex	A shorthand property for the flex-grow, flex-shrink, and the flex-basis
flex-basis	Specifies the initial length of a flex item
flex-grow	Specifies how much a flex item will grow relative to the rest of the flex the same container
flex-shrink	Specifies how much a flex item will shrink relative to the rest of the fle the same container
order	Specifies the order of the flex items inside the same container

CSS Grid Layout Module

Grid Layout

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

Grid Elements

A grid layout consists of a parent element, with one or more child elements.

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

```
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  background-color: #2196F3;
  padding: 10px;
}
.grid-item {
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>Grid Elements</h1>

<p>A Grid Layout must have a parent element with the <em>display</em> property set to
<em>grid</em> or <em>inline-grid</em>.</p>

<p>Direct child element(s) of the grid container automatically becomes grid items.</p>

<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  <div class="grid-item">8</div>
  <div class="grid-item">9</div>
</div>

</body>
</html>
```

Display Property

An HTML element becomes a grid container when its `display` property is set to `grid` or `inline-grid`.

```html
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: inline-grid;
  grid-template-columns: auto auto auto;
  background-color: #2196F3;
  padding: 10px;
}

.grid-item {
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>display: inline-grid</h1>

<p>Use display: inline-grid; to make an inline grid container:</p>

<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  <div class="grid-item">8</div>
  <div class="grid-item">9</div>
</div>

</body>
</html>
```

All direct children of the grid container automatically become grid items.

Grid Columns

The vertical lines of grid items are called columns.

Grid Rows

The horizontal lines of grid items are called rows.

Grid Gaps

The spaces between each column/row are called gaps.

You can adjust the gap size by using one of the following properties:

- `column-gap`
- `row-gap`
- `gap`

Example

The `column-gap` property sets the gap between the columns:

```
<!DOCTYPE html>

<html>

<head>

<style>

.grid-container {

  display: grid;

  column-gap: 50px;

  grid-template-columns: auto auto auto;

  background-color: #2196F3;

  padding: 10px;

}
```

```
.grid-item {
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>The column-gap Property</h1>

<p>Use the <em>column-gap</em> property to adjust the space
between the columns:</p>

<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  <div class="grid-item">8</div>
  <div class="grid-item">9</div>
</div>


</body>
</html>
```

Example

The `gap` property is a shorthand property for the `row-gap` and the `column-gap` properties:

The `gap` property can also be used to set both the row gap and the column gap in one value:

The `gap` property can also be used to set both the row gap and the column gap in one value:

All CSS Grid Properties

Property	Description
column-gap	Specifies the gap between the columns
gap	A shorthand property for the row-gap and the column-gap properties
grid	A shorthand property for the grid-template-rows, grid-template-columns, grid-template-areas, grid-auto-rows, grid-auto-columns, and the grid-auto-flow properties
grid-area	Either specifies a name for the grid item, or this property is a shorthand property for the grid-row-start, grid-column-start, grid-row-end, and grid-column-end properties

grid-auto-columns	Specifies a default column size
grid-auto-flow	Specifies how auto-placed items are inserted in the grid
grid-auto-rows	Specifies a default row size
grid-column	A shorthand property for the grid-column-start and the grid-column-end properties
grid-column-end	Specifies where to end the grid item
grid-column-gap	Specifies the size of the gap between columns
grid-column-start	Specifies where to start the grid item
grid-gap	A shorthand property for the grid-row-gap and grid-column-gap properties
grid-row	A shorthand property for the grid-row-start and the grid-row-end properties
grid-row-end	Specifies where to end the grid item
grid-row-gap	Specifies the size of the gap between rows
grid-row-start	Specifies where to start the grid item
grid-template	A shorthand property for the grid-template-rows, grid-template-columns and grid-areas properties
grid-template-areas	Specifies how to display columns and rows, using named grid items
grid-template-columns	Specifies the size of the columns, and how many columns in a grid layout
grid-template-rows	Specifies the size of the rows in a grid layout
row-gap	Specifies the gap between the grid rows

Grid Container

To make an HTML element behave as a grid container, you have to set the `display` property to `grid` or `inline-grid`.

Grid containers consist of grid items, placed inside columns and rows.

```
<!DOCTYPE html>

<html>

<head>

<style>

.grid-container {

  display: grid;

  grid-template-columns: auto auto auto auto;

  gap: 10px;

  background-color: #2196F3;

  padding: 10px;

}

.grid-container > div {

  background-color: rgba(255, 255, 255, 0.8);

  border: 1px solid black;

  text-align: center;

  font-size: 30px;

}

</style>

</head>

<body>

<h1>Grid Container</h1>
```

```
<p>A Grid Container consists of grid items arranged in columns and rows</p>

<div class="grid-container">

  <div>1</div>

  <div>2</div>

  <div>3</div>

  <div>4</div>

  <div>5</div>

  <div>6</div>

  <div>7</div>

  <div>8</div>

</div>

<p>Direct child elements(s) of the grid container automatically becomes grid items.</p>

</body>

</html>
```

The grid-template-columns Property

The `grid-template-columns` property defines the number of columns in your grid layout, and it can define the width of each column.

The value is a space-separated-list, where each value defines the width of the respective column.

If you want your grid layout to contain 4 columns, specify the width of the 4 columns, or "auto" if all columns should have the same width.

Example

Make a grid with 4 columns:

```
<!DOCTYPE html>

<html>

<head>

<style>

.grid-container {

  display: grid;

  grid-template-columns: auto auto auto auto;

  gap: 10px;

  background-color: #2196F3;

  padding: 10px;

}

.grid-container > div {

  background-color: rgba(255, 255, 255, 0.8);

  text-align: center;

  padding: 20px 0;
  font-size: 30px;

}

</style>

</head>

<body>

<h1>The grid-template-columns Property</h1>

<p>You can use the <em>grid-template-columns</em> property to specify
the number of columns in your grid layout.</p>

<div class="grid-container">

  <div>1</div>

  <div>2</div>

  <div>3</div>

  <div>4</div>

  <div>5</div>

  <div>6</div>

  <div>7</div>

  <div>8</div>

</div>
```

```
</body>
```

```
</html>
```

Note: If you have more than 4 items in a 4 columns grid, the grid will automatically add a new row to put the items in.

The `grid-template-columns` property can also be used to specify the size (width) of the columns.

Example

Set a size for the 4 columns:

```
.grid-container {
  display: grid;
  grid-template-columns: 80px 200px auto 40px;
}
```

The grid-template-rows Property

The `grid-template-rows` property defines the height of each row.

The value is a space-separated-list, where each value defines the height of the respective row:

Example

```
.grid-container {
  display: grid;
  grid-template-rows: 80px 200px;
}
```

The justify-content Property

The `justify-content` property is used to align the whole grid inside the container.

Note: The grid's total width has to be less than the container's width for the `justify-content` property to have any effect.

```
<!DOCTYPE html>

<html>

<head>

<style>

.grid-container {

  display: grid;

  justify-content: space-evenly;

  grid-template-columns: 50px 50px 50px; /*Make the grid smaller than the container*/

  gap: 10px;

  background-color: #2196F3;

  padding: 10px;

}

.grid-container > div {

  background-color: rgba(255, 255, 255, 0.8);

  text-align: center;

  padding: 20px 0;

  font-size: 30px;

}

</style>

</head>

<body>

<h1>The justify-content Property</h1>
```

```
<p>Use the <em>justify-content</em> property to align the grid inside the container.</p>

<p>The value "space-evenly" will give the columns equal amount of space between, and around them:</p>

<div class="grid-container">

  <div>1</div>

  <div>2</div>

  <div>3</div>

  <div>4</div>

  <div>5</div>

  <div>6</div>

</div>

</body>

</html>
```

```
.grid-container {
  display: grid;
  justify-content: space-around;
}
```

```
.grid-container {
  display: grid;
  justify-content: space-between;
}
```

```
.grid-container {
  display: grid;
  justify-content: center;
}
```

```
.grid-container {
  display: grid;
  justify-content: start;
}
```

```
.grid-container {
  display: grid;
  justify-content: end;
}
```

The align-content Property

The `align-content` property is used to vertically align the whole grid inside the container.

Note: The grid's total height has to be less than the container's height for the `align-content` property to have any effect.

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: center;
}
```

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: space-evenly;
}
```

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: space-around;
}
```

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: space-between;
}
```

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: start;
}
```

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: end;
}
```

CSS Grid Item

Child Elements (Items)

A grid container contains grid items.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

The grid-column Property:

The `grid-column` property defines on which column(s) to place an item.

You define where the item will start, and where the item will end.

Note: The `grid-column` property is a shorthand property for the `grid-column-start` and the `grid-column-end` properties.

To place an item, you can refer to line numbers, or use the keyword "span" to define how many columns the item will span.

Example

Make "item1" start on column 1 and end before column 5:

```
<!DOCTYPE html>

<html>

<head>

<style>

.grid-container {

  display: grid;
  grid-template-columns: auto auto auto auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}

.item1 {
  grid-column: 1 / 5;
}
</style>
</head>
<body>

<h1>The grid-column Property</h1>

<p>Use the <em>grid-column</em> property to specify where to place
an item.</p>
<p>Item1 will start on column 1 and end before column 5:</p>

<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
  <div class="item9">9</div>
  <div class="item10">10</div>
  <div class="item11">11</div>
  <div class="item12">12</div>
  <div class="item13">13</div>
  <div class="item14">14</div>
  <div class="item15">15</div>
</div>

</body>
</html>
```

Example

Make "item1" start on column 1 and span 3 columns:

```
.item1 {
  grid-column: 1 / span 3;
}
```

Example

Make "item2" start on column 2 and span 3 columns:

```
.item2 {
  grid-column: 2 / span 3;
}
```

The grid-row Property:

The `grid-row` property defines on which row to place an item.

You define where the item will start, and where the item will end.

Note: The `grid-row` property is a shorthand property for the `grid-row-start` and the `grid-row-end` properties.

To place an item, you can refer to line numbers, or use the keyword "span" to define how many rows the item will span:

Example

Make "item1" start on row-line 1 and end on row-line 4:

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}

.item1 {
  grid-row: 1 / 4;
}
</style>
</head>
<body>

<h1>The grid-row Property</h1>

<p>Use the <em>grid-row</em> property to specify where to place an
item.</p>
<p>Item1 will start on row-line 1 and end on row-line 4:</p>

<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
  <div class="item9">9</div>
  <div class="item10">10</div>
  <div class="item11">11</div>
  <div class="item12">12</div>
  <div class="item13">13</div>
  <div class="item14">14</div>
  <div class="item15">15</div>
  <div class="item16">16</div>
```

```
</div>


</body>

</html>
```

The grid-area Property

The `grid-area` property can be used as a shorthand property for the `grid-row-start`, `grid-column-start`, `grid-row-end` and the `grid-column-end` properties.

Example

Make "item8" start on row-line 1 and column-line 2, and end on row-line 5 and column line 6:

```
.item8 {
  grid-area: 1 / 2 / 5 / 6;
}
```

Example

Make "item8" start on row-line 2 and column-line 1, and span 2 rows and 3 columns:

```
.item8 {
  grid-area: 2 / 1 / span 2 / span 3;
}
```

Naming Grid Items

The `grid-area` property can also be used to assign names to grid items.

Named grid items can be referred to by the `grid-template-areas` property of the grid container.

Example

Item1 gets the name "myArea" and spans all five columns in a five columns grid layout:

```
.item1 {
  grid-area: myArea;
}
.grid-container {
  grid-template-areas: 'myArea myArea myArea myArea myArea';
}
```

Each row is defined by apostrophes (' ')

The columns in each row is defined inside the apostrophes, separated by a space.

Note: A period sign represents a grid item with no name.

Example

Let "myArea" span two columns in a five columns grid layout (period signs represent items with no name):

```
.item1 {
  grid-area: myArea;
}
.grid-container {
  grid-template-areas: 'myArea myArea . . .';
}
```

To define two rows, define the column of the second row inside another set of apostrophes:

Example

Make "item1" span two columns and two rows:

```
.grid-container {
  grid-template-areas: 'myArea myArea . . .' 'myArea myArea . . .';
}
```

Example

Name all items, and make a ready-to-use webpage template:

```
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }

.grid-container {
  grid-template-areas:
    'header header header header header header'
    'menu main main main right right'
    'menu footer footer footer footer footer';
}
```

The Order of the Items

The Grid Layout allows us to position the items anywhere we like.

The first item in the HTML code does not have to appear as the first item in the grid.

```
.item1 { grid-area: 1 / 3 / 2 / 4; }
.item2 { grid-area: 2 / 3 / 3 / 4; }
.item3 { grid-area: 1 / 1 / 2 / 2; }
.item4 { grid-area: 1 / 2 / 2 / 3; }
.item5 { grid-area: 2 / 1 / 3 / 2; }
.item6 { grid-area: 2 / 2 / 3 / 3; }
```

You can re-arrange the order for certain screen sizes, by using media queries:

Example

```
@media only screen and (max-width: 500px) {
  .item1 { grid-area: 1 / span 3 / 2 / 4; }
  .item2 { grid-area: 3 / 3 / 4 / 4; }
  .item3 { grid-area: 2 / 1 / 3 / 2; }
  .item4 { grid-area: 2 / 2 / span 2 / 3; }
  .item5 { grid-area: 3 / 1 / 4 / 2; }
  .item6 { grid-area: 2 / 3 / 3 / 4; }
}
```

Week 5-6: Forms and Media
Week 5:

HTML Forms

The <form> Element

An HTML form is used to collect user input. The user input is most often sent to a server for processing.The **<form>** element defines an HTML form:

HTML forms contain **form elements**.

Form elements are different types of input elements, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

The HTML `<input>` element is the most used form element.

An `<input>` element can be displayed in many ways, depending on the `type` attribute.

Here are some examples:

Type	Description
<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="checkbox">	Displays a checkbox (for selecting zero or more of many choices)
<input type="submit">	Displays a submit button (for submitting the form)
<input type="button">	Displays a clickable button

Text Fields

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

Example

A form with input fields for text:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

Note: The form itself is not visible. Also note that the default width of an input field is 20 characters.

The <label> Element

Notice the use of the `<label>` element in the example above.

The `<label>` tag defines a label for many form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focuses on the input element.

The `<label>` element also helps users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.

The `for` attribute of the `<label>` tag should be equal to the `id` attribute of the `<input>` element to bind them together.

Radio Buttons

The `<input type="radio">` defines a radio button.

Radio buttons let a user select ONE of a limited number of choices.

Example

A form with radio buttons:

```
<p>Choose your favorite Web language:</p>

<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaSc
ript">
  <label for="javascript">JavaScript</label>
</form>
```

Checkboxes

The `<input type="checkbox">` defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

Example

A form with checkboxes:

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```

The Submit Button

The `<input type="submit">` defines a button for submitting the form data to a form-handler.

The form-handler is typically a file on the server with a script for processing input data.

The form-handler is specified in the form's `action` attribute.

Example

A form with a submit button:

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

The Name Attribute for <input>

Notice that each input field must have a `name` attribute to be submitted.

If the `name` attribute is omitted, the value of the input field will not be sent at all.

Example

This example will not submit the value of the "First name" input field:

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" value="John"><br><br>
  <input type="submit" value="Submit">
</form>
```

Exercise:

In the form below, add an input field with the type "button" and the value "OK".

```
<form>
<>
</form>
```

HTML Form Attributes

This chapter describes the different attributes for the HTML `<form>` element.

The Action Attribute

The `action` attribute defines the action to be performed when the form is submitted.

Usually, the form data is sent to a file on the server when the user clicks on the submit button.

In the example below, the form data is sent to a file called "action_page.php". This file contains a server-side script that handles the form data:

Example

On submit, send form data to "action_page.php":

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Tip: If the `action` attribute is omitted, the action is set to the current page.

The Target Attribute

The `target` attribute specifies where to display the response that is received after submitting the form.

The `target` attribute can have one of the following values:

Value	Description
_blank	The response is displayed in a new window or tab
_self	The response is displayed in the current window
_parent	The response is displayed in the parent frame
_top	The response is displayed in the full body of the window
framename	The response is displayed in a named iframe

The default value is `_self` which means that the response will open in the current window.

Example

Here, the submitted result will open in a new browser tab:

```
<form action="/action_page.php" target="_blank">
```

The Method Attribute

The `method` attribute specifies the HTTP method to be used when submitting the form data.

The form-data can be sent as URL variables (with `method="get"`) or as HTTP post transaction (with `method="post"`).

The default HTTP method when submitting form data is GET.

Example

This example uses the GET method when submitting the form data:

```
<form action="/action_page.php" method="get">
```

Example

This example uses the POST method when submitting the form data:

```
<form action="/action_page.php" method="post">
```

Notes on GET:

- Appends the form data to the URL, in name/value pairs
- NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
- The length of a URL is limited (2048 characters)
- Useful for form submissions where a user wants to bookmark the result
- GET is good for non-secure data, like query strings in Google

Notes on POST:

- Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)

- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

Tip: Always use POST if the form data contains sensitive or personal information!

The Autocomplete Attribute

The `autocomplete` attribute specifies whether a form should have autocomplete on or off.

When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

Example

A form with autocomplete on:

```
<form action="/action_page.php" autocomplete="on">
```

The Novalidate Attribute

The `novalidate` attribute is a boolean attribute.

When present, it specifies that the form-data (input) should not be validated when submitted.

Example

A form with a novalidate attribute:

```
<form action="/action_page.php" novalidate>
```

Exercise:

Add a submit button, and specify that the form should go to "/action_page.php".

```
<form ="/action_page.php">
Name: <input type="text" name="name">
<>
</form>
```

List of All <form> Attributes

Attribute	Description
accept-charset	Specifies the character encodings used for form submission
action	Specifies where to send the form-data when a form is submitted
autocomplete	Specifies whether a form should have autocomplete on or off
enctype	Specifies how the form-data should be encoded when submitting it to the server (only for method="post")
method	Specifies the HTTP method to use when sending form-data
name	Specifies the name of the form
novalidate	Specifies that the form should not be validated when submitted
rel	Specifies the relationship between a linked resource and the current document
target	Specifies where to display the response that is received after submitting the form

HTML Form Elements

This chapter describes all the different HTML form elements.

The HTML <form> Elements

The HTML `<form>` element can contain one or more of the following form elements:

- `<input>`
- `<label>`
- `<select>`
- `<textarea>`
- `<button>`
- `<fieldset>`
- `<legend>`
- `<datalist>`
- `<output>`
- `<option>`
- `<optgroup>`

The <input> Element

One of the most used form elements is the `<input>` element.

The `<input>` element can be displayed in several ways, depending on the `type` attribute.

Example

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

The <label> Element

The `<label>` element defines a label for several form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.

The `<label>` element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.

The `for` attribute of the `<label>` tag should be equal to the `id` attribute of the `<input>` element to bind them together.

The <select> Element

The `<select>` element defines a drop-down list:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The `<option>` element defines an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the `selected` attribute to the option:

Example

```
<option value="fiat" selected>Fiat</option>
```

Visible Values:

Use the `size` attribute to specify the number of visible values:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="3">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

Allow Multiple Selections:

Use the `multiple` attribute to allow the user to select more than one value:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="4" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The <textarea> Element

The `<textarea>` element defines a multi-line input field (a text area):

Example

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The `rows` attribute specifies the visible number of lines in a text area.

The `cols` attribute specifies the visible width of a text area.

This is how the HTML code above will be displayed in a browser:

You can also define the size of the text area by using CSS:

Example

```
<textarea name="message" style="width:200px; height:600px;">
The cat was playing in the garden.
</textarea>
```

The <button> Element

The `<button>` element defines a clickable button:

Example

```
<button type="button" onclick="alert('Hello World!')">Click
Me!</button>
```

Note: Always specify the `type` attribute for the button element. Different browsers may use different default types for the button element.

The <fieldset> and <legend> Elements

The `<fieldset>` element is used to group related data in a form.

The `<legend>` element defines a caption for the `<fieldset>` element.

Example

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname" value="John"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname" value="Doe"><br><br>
    <input type="submit" value="Submit">
  </fieldset>
</form>
```

The <datalist> Element

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

Users will see a drop-down list of the pre-defined options as they input data.

The `list` attribute of the `<input>` element, must refer to the `id` attribute of the `<datalist>` element.

Example

```
<form action="/action_page.php">
  <input list="browsers">
  <datalist id="browsers">
    <option value="Edge">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
</form>
```

The <output> Element

The `<output>` element represents the result of a calculation (like one performed by a script).

Example

Perform a calculation and show the result in an `<output>` element:

```
<form action="/action_page.php"
  oninput="x.value=parseInt(a.value)+parseInt(b.value)">
  0
  <input type="range"  id="a" name="a" value="50">
  100 +
  <input type="number" id="b" name="b" value="50">
  =
  <output name="x" for="a b"></output>
  <br><br>
  <input type="submit">
</form>
```

Exercise:

In the form below, add an empty drop down list with the name "cars".

```
<form action="/action_page.php">
<>
</>
</form>
```

HTML Form Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element
<fieldset>	Groups related elements in a form
<legend>	Defines a caption for a <fieldset> element
<select>	Defines a drop-down list
<optgroup>	Defines a group of related options in a drop-down list
<option>	Defines an option in a drop-down list
<button>	Defines a clickable button
<datalist>	Specifies a list of pre-defined options for input controls
<output>	Defines the result of a calculation

HTML Input Types

This chapter describes the different types for the HTML `<input>` element.

HTML Input Types

Here are the different input types you can use in HTML:

- `<input type="button">`
- `<input type="checkbox">`
- `<input type="color">`
- `<input type="date">`
- `<input type="datetime-local">`
- `<input type="email">`
- `<input type="file">`
- `<input type="hidden">`
- `<input type="image">`
- `<input type="month">`
- `<input type="number">`
- `<input type="password">`
- `<input type="radio">`
- `<input type="range">`
- `<input type="reset">`
- `<input type="search">`
- `<input type="submit">`
- `<input type="tel">`
- `<input type="text">`
- `<input type="time">`
- `<input type="url">`
- `<input type="week">`

Tip: The default value of the `type` attribute is "text".

Input Type Text

`<input type="text">` defines a **single-line text input field**:

Example

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

Input Type Password

`<input type="password">` defines a **password field**:

Example

```
<form>
  <label for="username">Username:</label><br>
  <input type="text" id="username" name="username"><br>
  <label for="pwd">Password:</label><br>
  <input type="password" id="pwd" name="pwd">
</form>
```

The characters in a password field are masked (shown as asterisks or circles).

Input Type Submit

`<input type="submit">` defines a button for **submitting** form data to a **form-handler**.

The form-handler is typically a server page with a script for processing input data.

The form-handler is specified in the form's `action` attribute:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

If you omit the submit button's value attribute, the button will get a default text:

Input Type Reset

`<input type="reset">` defines a **reset button** that will reset all form values to their default values:

Example

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
  <input type="reset" value="Reset">
</form>
```

If you change the input values and then click the "Reset" button, the form-data will be reset to the default values.

Input Type Radio

`<input type="radio">` defines a **radio button**.

Radio buttons let a user select ONLY ONE of a limited number of choices:

Example

```
<p>Choose your favorite Web language:</p>

<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaSc
ript">
  <label for="javascript">JavaScript</label>
</form>
```

Input Type Checkbox

`<input type="checkbox">` defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

Example

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```

Input Type Button

`<input type="button">` defines a **button**:

Example

```
<input type="button" onclick="alert('Hello World!')" value="Click Me!">
```

Input Type Color

The `<input type="color">` is used for input fields that should contain a color.

Depending on browser support, a color picker can show up in the input field.

Example

```
<form>
  <label for="favcolor">Select your favorite color:</label>
  <input type="color" id="favcolor" name="favcolor">
</form>
```

Input Type Date

The `<input type="date">` is used for input fields that should contain a date.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="birthday">Birthday:</label>
  <input type="date" id="birthday" name="birthday">
</form>
```

You can also use the `min` and `max` attributes to add restrictions to dates:

Example

```
<form>
  <label for="datemax">Enter a date before 1980-01-01:</label>
  <input type="date" id="datemax" name="datemax" max="1979-12-
31"><br><br>
  <label for="datemin">Enter a date after 2000-01-01:</label>
  <input type="date" id="datemin" name="datemin" min="2000-01-02">
</form>
```

Input Type Datetime-local

The `<input type="datetime-local">` specifies a date and time input field, with no time zone.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="birthdaytime">Birthday (date and time):</label>
  <input type="datetime-local" id="birthdaytime" name="birthdaytime">
</form>
```

Input Type Email

The `<input type="email">` is used for input fields that should contain an e-mail address.

Depending on browser support, the e-mail address can be automatically validated when submitted.

Some smartphones recognize the email type, and add ".com" to the keyboard to match email input.

Example

```
<form>
  <label for="email">Enter your email:</label>
  <input type="email" id="email" name="email">
</form>
```

Input Type Image

The `<input type="image">` defines an image as a submit button.

The path to the image is specified in the `src` attribute.

Example

```
<form>
<input type="image" src="img_submit.gif" alt="Submit" width="48" height
="48">
</form>
```

Input Type File

The `<input type="file">` defines a file-select field and a "Browse" button for file uploads.

Example

```
<form>
  <label for="myfile">Select a file:</label>
  <input type="file" id="myfile" name="myfile">
</form>
```

Input Type Hidden

The `<input type="hidden">` defines a hidden input field (not visible to a user).

A hidden field lets web developers include data that cannot be seen or modified by users when a form is submitted.

A hidden field often stores what database record that needs to be updated when the form is submitted.

Note: While the value is not displayed to the user in the page's content, it is visible (and can be edited) using any browser's developer tools or "View Source" functionality. Do not use hidden inputs as a form of security!

Example

```
<form>
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="hidden" id="custId" name="custId" value="3487">
  <input type="submit" value="Submit">
</form>
```

Input Type Month

The `<input type="month">` allows the user to select a month and year.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="bdaymonth">Birthday (month and year):</label>
  <input type="month" id="bdaymonth" name="bdaymonth">
</form>
```

Input Type Number

The `<input type="number">` defines a **numeric** input field.

You can also set restrictions on what numbers are accepted.

The following example displays a numeric input field, where you can enter a value from 1 to 5:

Example

```
<form>
  <label for="quantity">Quantity (between 1 and 5):</label>
  <input type="number" id="quantity" name="quantity" min="1" max="5">
</form>
```

Input Restrictions

Here is a list of some common input restrictions:

Attribute	Description
checked	Specifies that an input field should be pre-selected when the page loads (for type="checkbox" or type="radio")
disabled	Specifies that an input field should be disabled
max	Specifies the maximum value for an input field
maxlength	Specifies the maximum number of character for an input field
min	Specifies the minimum value for an input field
pattern	Specifies a regular expression to check the input value against
readonly	Specifies that an input field is read only (cannot be changed)
required	Specifies that an input field is required (must be filled out)
size	Specifies the width (in characters) of an input field
step	Specifies the legal number intervals for an input field
value	Specifies the default value for an input field

You will learn more about input restrictions in the next chapter.

The following example displays a numeric input field, where you can enter a value from 0 to 100, in steps of 10. The default value is 30:

Example

```
<form>
  <label for="quantity">Quantity:</label>
  <input type="number" id="quantity" name="quantity" min="0" max="100"
step="10" value="30">
</form>
```

Input Type Range

The `<input type="range">` defines a control for entering a number whose exact value is not important (like a slider control). Default range is 0 to 100. However, you can set restrictions on what numbers are accepted with the `min`, `max`, and `step` attributes:

Example

```
<form>
  <label for="vol">Volume (between 0 and 50):</label>
  <input type="range" id="vol" name="vol" min="0" max="50">
</form>
```

Input Type Search

The `<input type="search">` is used for search fields (a search field behaves like a regular text field).

Example

```
<form>
  <label for="gsearch">Search Google:</label>
  <input type="search" id="gsearch" name="gsearch">
</form>
```

Input Type Tel

The `<input type="tel">` is used for input fields that should contain a telephone number.

Example

```
<form>
  <label for="phone">Enter your phone number:</label>
  <input type="tel" id="phone" name="phone" pattern="[0-9]{3}-[0-9]{2}-
```

```
[0-9]{3}">
</form>
```

Input Type Time

The `<input type="time">` allows the user to select a time (no time zone).

Depending on browser support, a time picker can show up in the input field.

Example

```
<form>
  <label for="appt">Select a time:</label>
  <input type="time" id="appt" name="appt">
</form>
```

Input Type Url

The `<input type="url">` is used for input fields that should contain a URL address.

Depending on browser support, the url field can be automatically validated when submitted.

Some smartphones recognize the url type, and adds ".com" to the keyboard to match url input.

```
<form>
  <label for="homepage">Add your homepage:</label>
  <input type="url" id="homepage" name="homepage">
</form>
```

Input Type Week

The `<input type="week">` allows the user to select a week and year.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="week">Select a week:</label>
  <input type="week" id="week" name="week">
</form>
```

Exercise:

In the form below, add an input field for text, with the name "username" .

```
<form action="/action_page.php">
<>
</form>
```

HTML Input Type Attribute

Tag	Description
<input type="">	Specifies the input type to display

HTML Input Attributes

This chapter describes the different attributes for the HTML `<input>` element.

The value Attribute

The input `value` attribute specifies an initial value for an input field:

Example

Input fields with initial (default) values:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe">
</form>
```

The readonly Attribute

The input `readonly` attribute specifies that an input field is read-only.

A read-only input field cannot be modified (however, a user can tab to it, highlight it, and copy the text from it).

The value of a read-only input field will be sent when submitting the form!

A read-only input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John" readonly><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe">
</form>
```

The disabled Attribute

The input `disabled` attribute specifies that an input field should be disabled.

A disabled input field is unusable and un-clickable.

The value of a disabled input field will not be sent when submitting the form!

A disabled input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John" disabled><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe">
</form>
```

The size Attribute

The input `size` attribute specifies the visible width, in characters, of an input field.

The default value for `size` is 20.

Note: The `size` attribute works with the following input types: text, search, tel, url, email, and password.

Set a width for an input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" size="50"><br>
  <label for="pin">PIN:</label><br>
  <input type="text" id="pin" name="pin" size="4">
</form>
```

The maxlength Attribute

The input `maxlength` attribute specifies the maximum number of characters allowed in an input field.

Note: When a `maxlength` is set, the input field will not accept more than the specified number of characters. However, this attribute does not provide any feedback. So, if you want to alert the user, you must write JavaScript code.

Example

Set a maximum length for an input field:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" size="50"><br>
  <label for="pin">PIN:</label><br>
  <input type="text" id="pin" name="pin" maxlength="4" size="4">
</form>
```

The min and max Attributes

The input `min` and `max` attributes specify the minimum and maximum values for an input field.

The `min` and `max` attributes work with the following input types: number, range, date, datetime-local, month, time and week.

Tip: Use the max and min attributes together to create a range of legal values.

Set a max date, a min date, and a range of legal values:

```
<form>
  <label for="datemax">Enter a date before 1980-01-01:</label>
  <input type="date" id="datemax" name="datemax" max="1979-12-
31"><br><br>

  <label for="datemin">Enter a date after 2000-01-01:</label>
  <input type="date" id="datemin" name="datemin" min="2000-01-
02"><br><br>

  <label for="quantity">Quantity (between 1 and 5):</label>
  <input type="number" id="quantity" name="quantity" min="1" max="5">
</form>
```

The multiple Attribute

The input `multiple` attribute specifies that the user is allowed to enter more than one value in an input field.

The `multiple` attribute works with the following input types: email, and file.

A file upload field that accepts multiple values:

```
<form>
  <label for="files">Select files:</label>
  <input type="file" id="files" name="files" multiple>
</form>
```

The pattern Attribute

The input `pattern` attribute specifies a regular expression that the input field's value is checked against, when the form is submitted.

The `pattern` attribute works with the following input types: text, date, search, url, tel, email, and password.

Tip: Use the global title attribute to describe the pattern to help the user.

The placeholder Attribute

The input `placeholder` attribute specifies a short hint that describes the expected value of an input field (a sample value or a short description of the expected format).

The short hint is displayed in the input field before the user enters a value.

The `placeholder` attribute works with the following input types: text, search, url, tel, email, and password.

An input field with a placeholder text:

```
<form>
  <label for="phone">Enter a phone number:</label>
  <input type="tel" id="phone" name="phone"
  placeholder="123-45-678"
  pattern="[0-9]{3}-[0-9]{2}-[0-9]{3}">
</form>
```

The required Attribute

The input `required` attribute specifies that an input field must be filled out before submitting the form.

The `required` attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

A required input field:

```html
<form>
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required>
</form>
```

The step Attribute

The input `step` attribute specifies the legal number intervals for an input field.

Example: if step="3", legal numbers could be -3, 0, 3, 6, etc.

Tip: This attribute can be used together with the max and min attributes to create a range of legal values.

The `step` attribute works with the following input types: number, range, date, datetime-local, month, time and week.

Example

An input field with a specified legal number intervals:

```html
<form>
  <label for="points">Points:</label>
  <input type="number" id="points" name="points" step="3">
</form>
```

Note: Input restrictions are not foolproof, and JavaScript provides many ways to add illegal input. To safely restrict input, it must also be checked by the receiver (the server)!

The autofocus Attribute

The input `autofocus` attribute specifies that an input field should automatically get focus when the page loads.

Example

Let the "First name" input field automatically get focus when the page loads:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" autofocus><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

The height and width Attributes

The input `height` and `width` attributes specify the height and width of an `<input type="image">` element.

Tip: Always specify both the height and width attributes for images. If height and width are set, the space required for the image is reserved when the page is loaded. Without these attributes, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

Example

Define an image as the submit button, with height and width attributes:

```
<form>
  <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>
  <input type="image" src="img_submit.gif" alt="Submit" width="48"
height="48">
</form>
```

The list Attribute

The input `list` attribute refers to a `<datalist>` element that contains pre-defined options for an <input> element.

Example

An <input> element with pre-defined values in a <datalist>:

```
<form>
  <input list="browsers">
  <datalist id="browsers">
    <option value="Edge">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
</form>
```

The autocomplete Attribute

The input `autocomplete` attribute specifies whether a form or an input field should have autocomplete on or off.

Autocomplete allows the browser to predict the value. When a user starts to type in a field, the browser should display options to fill in the field, based on earlier typed values.

The `autocomplete` attribute works with `<form>` and the following `<input>` types: text, search, url, tel, email, password, datepickers, range, and color.

Example

An HTML form with autocomplete on, and off for one input field:

```
<form action="/action_page.php" autocomplete="on">
  <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" autocomplete="off"><br><b
r>
  <input type="submit" value="Submit">
</form>
```

Tip: In some browsers you may need to activate an autocomplete function for this to work (Look under "Preferences" in the browser's menu).

Exercise:

In the input field below, add placeholder that says "Your name here".

```
<form action="/action_page.php">
<input type="text" >
</form>
```

HTML Form and Input Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control

HTML Input form* Attributes

This chapter describes the different `form*` attributes for the HTML `<input>` element.

The form Attribute

The input `form` attribute specifies the form the `<input>` element belongs to.

The value of this attribute must be equal to the id attribute of the <form> element it belongs to.

Example

An input field located outside of the HTML form (but still a part of the form):

```
<form action="/action_page.php" id="form1">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="submit" value="Submit">
</form>

<label for="lname">Last name:</label>
<input type="text" id="lname" name="lname" form="form1">
```

The formaction Attribute

The input `formaction` attribute specifies the URL of the file that will process the input when the form is submitted.

Note: This attribute overrides the `action` attribute of the `<form>` element.

The `formaction` attribute works with the following input types: submit and image.

Example

An HTML form with two submit buttons, with different actions:

```
<form action="/action_page.php">
  <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>
  <input type="submit" value="Submit">
  <input type="submit" formaction="/action_page2.php" value="Submit as
Admin">
</form>
```

The formenctype Attribute

The input `formenctype` attribute specifies how the form-data should be encoded when submitted (only for forms with method="post").

Note: This attribute overrides the enctype attribute of the `<form>` element.

The `formenctype` attribute works with the following input types: submit and image.

Example

A form with two submit buttons. The first sends the form-data with default encoding, the second sends the form-data encoded as "multipart/form-data":

```
<form action="/action_page_binary.asp" method="post">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <input type="submit" value="Submit">
  <input type="submit" formenctype="multipart/form-data"
  value="Submit as Multipart/form-data">
</form>
```

The formmethod Attribute

The input `formmethod` attribute defines the HTTP method for sending form-data to the action URL.

Note: This attribute overrides the method attribute of the `<form>` element.

The `formmethod` attribute works with the following input types: submit and image.

The form-data can be sent as URL variables (method="get") or as an HTTP post transaction (method="post").

Notes on the "get" method:

- This method appends the form-data to the URL in name/value pairs
- This method is useful for form submissions where a user want to bookmark the result
- There is a limit to how much data you can place in a URL (varies between browsers), therefore, you cannot be sure that all of the form-data will be correctly transferred

- Never use the "get" method to pass sensitive information! (password or other sensitive information will be visible in the browser's address bar)

Notes on the "post" method:

- This method sends the form-data as an HTTP post transaction
- Form submissions with the "post" method cannot be bookmarked
- The "post" method is more robust and secure than "get", and "post" does not have size limitations

Example

A form with two submit buttons. The first sends the form-data with method="get". The second sends the form-data with method="post":

```
<form action="/action_page.php" method="get">
  <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>
  <input type="submit" value="Submit using GET">
  <input type="submit" formmethod="post" value="Submit using POST">
</form>
```

The formtarget Attribute

The input `formtarget` attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.

Note: This attribute overrides the target attribute of the `<form>` element.

The `formtarget` attribute works with the following input types: submit and image.

Example

A form with two submit buttons, with different target windows

```
<form action="/action_page.php">
  <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>
    <input type="submit" value="Submit">
    <input type="submit" formtarget="_blank" value="Submit to a new
  window/tab">
  </form>
```

The formnovalidate Attribute

The input `formnovalidate` attribute specifies that an <input> element should not be validated when submitted.

Note: This attribute overrides the novalidate attribute of the `<form>` element.

The `formnovalidate` attribute works with the following input types: submit.

Example

A form with two submit buttons (with and without validation):

```
<form action="/action_page.php">
  <label for="email">Enter your email:</label>
  <input type="email" id="email" name="email"><br><br>
  <input type="submit" value="Submit">
  <input type="submit" formnovalidate="formnovalidate"
  value="Submit without validation">
</form>
```

The novalidate Attribute

The `novalidate` attribute is a `<form>` attribute.

When present, novalidate specifies that all of the form-data should not be validated when submitted.

Example

Specify that no form-data should be validated on submit:

```
<form action="/action_page.php" novalidate>
  <label for="email">Enter your email:</label>
  <input type="email" id="email" name="email"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML Form and Input Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control

CSS Text & Fonts

CSS has a lot of properties for formatting text.

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  border: 1px solid gray;
  padding: 8px;
}
h1 {
  text-align: center;
  text-transform: uppercase;
  color: #4CAF50;
}

p {
  text-indent: 50px;
  text-align: justify;
  letter-spacing: 3px;
}

a {
  text-decoration: none;
  color: #008CBA;
}
</style>
</head>
<body>
<div>
  <h1>text formatting</h1>
  <p>This text is styled with some of the text formatting properties. The heading uses the text-align, text-transform, and color properties.
  The paragraph is indented, aligned, and the space between characters is specified. The underline is removed from this colored
  <a target="_blank" href="tryit.asp?filename=trycss_text">"Try it Yourself"</a> link.</p>
</div>

</body>
</html>
```

Text Color

The `color` property is used to set the color of the text. The color is specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

The default text color for a page is defined in the body selector.

```
body {
  color: blue;
}

h1 {
  color: green;
}
```

Text Color and Background Color

In this example, we define both the `background-color` property and the `color` property:

Example

```
body {
  background-color: lightgrey;
  color: blue;
}

h1 {
  background-color: black;
  color: white;
}

div {
  background-color: blue;
  color: white;
}
```

Important: High contrast is very important for people with vision problems. So, always ensure that the contrast between the text color and the background color (or background image) is good!

The CSS Text Color Property

Property	Description
color	Specifies the color of text

CSS Text Alignment

Text Alignment and Text Direction

In this chapter you will learn about the following properties:

- `text-align`
- `text-align-last`
- `direction`
- `unicode-bidi`
- `vertical-align`

Text Alignment

The `text-align` property is used to set the horizontal alignment of a text.

A text can be left or right aligned, centered, or justified.

The following example shows center aligned, and left and right aligned text (left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left):

Example

```
h1 {
  text-align: center;
}

h2 {
  text-align: left;
}

h3 {
  text-align: right;
}
```

When the `text-align` property is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers):

```
div {
  text-align: justify;
}
```

Text Align Last

The `text-align-last` property specifies how to align the last line of a text.

Example

Align the last line of text in three <p> elements:

```
p.a {
  text-align-last: right;
}

p.b {
  text-align-last: center;
}

p.c {
  text-align-last: justify;
}
```

Text Direction

The `direction` and `unicode-bidi` properties can be used to change the text direction of an element:

Example

```
p {
  direction: rtl;
  unicode-bidi: bidi-override;
}
```

```
<body>

<p>This is the default text direction.</p>

<p class="ex1">This is right-to-left text direction.</p>
```

```
</body>
```

Vertical Alignment

The `vertical-align` property sets the vertical alignment of an element.

Example

Set the vertical alignment of an image in a text:

```
<!DOCTYPE html>
<html>
<head>
<style>
img.a {
  vertical-align: baseline;
}

img.b {
  vertical-align: text-top;
}

img.c {
  vertical-align: text-bottom;
}

img.d {
  vertical-align: sub;
}

img.e {
  vertical-align: super;
}
</style>
</head>
<body>

<h1>The vertical-align Property</h1>

<h2>vertical-align: baseline (default):</h2>
<p>An <img class="a" src="sqpurple.gif" width="9" height="9"> image with a default
alignment.</p>

<h2>vertical-align: text-top:</h2>
<p>An <img class="b" src="sqpurple.gif" width="9" height="9"> image with a text-top
alignment.</p>

<h2>vertical-align: text-bottom:</h2>
<p>An <img class="c" src="sqpurple.gif" width="9" height="9"> image with a text-bottom
alignment.</p>

<h2>vertical-align: sub:</h2>
```

```
<p>An <img class="d" src="sqpurple.gif" width="9" height="9"> image with a sub
alignment.</p>

<h2>vertical-align: sup:</h2>
<p>An <img class="e" src="sqpurple.gif" width="9" height="9"> image with a super
alignment.</p>

</body>
</html>
```

The CSS Text Alignment/Direction Properties

Property	Description
direction	Specifies the text direction/writing direction
text-align	Specifies the horizontal alignment of text
text-align-last	Specifies how to align the last line of a text
unicode-bidi	Used together with the direction property to set or return whether the text should be overridden to support multiple languages in the same document
vertical-align	Sets the vertical alignment of an element

CSS Text Decoration

Text Decoration

In this chapter you will learn about the following properties:

- `text-decoration-line`
- `text-decoration-color`
- `text-decoration-style`
- `text-decoration-thickness`
- `text-decoration`

Add a Decoration Line to Text

The `text-decoration-line` property is used to add a decoration line to text.

Tip: You can combine more than one value, like overline and underline to display lines both over and under a text.

Example

```
h1 {
  text-decoration-line: overline;
}

h2 {
  text-decoration-line: line-through;
}

h3 {
  text-decoration-line: underline;
}

p {
  text-decoration-line: overline underline;
}
```

Note: It is not recommended to underline text that is not a link, as this often confuses the reader.

Specify a Color for the Decoration Line

The `text-decoration-color` property is used to set the color of the decoration line.

Example

```
h1 {
  text-decoration-line: overline;
  text-decoration-color: red;
}

h2 {
  text-decoration-line: line-through;
  text-decoration-color: blue;
}

h3 {
  text-decoration-line: underline;
  text-decoration-color: green;
}

p {
  text-decoration-line: overline underline;
  text-decoration-color: purple;
}
```

Specify a Style for the Decoration Line

The `text-decoration-style` property is used to set the style of the decoration line.

```
h1 {
  text-decoration-line: underline;
  text-decoration-style: solid;
}

h2 {
  text-decoration-line: underline;
  text-decoration-style: double;
}

h3 {
  text-decoration-line: underline;
  text-decoration-style: dotted;
}

p.ex1 {
  text-decoration-line: underline;
  text-decoration-style: dashed;
}

p.ex2 {
  text-decoration-line: underline;
  text-decoration-style: wavy;
}

p.ex3 {
  text-decoration-line: underline;
  text-decoration-color: red;
  text-decoration-style: wavy;
}
```

Specify the Thickness for the Decoration Line

The `text-decoration-thickness` property is used to set the thickness of the decoration line.

Example

```
h1 {
  text-decoration-line: underline;
  text-decoration-thickness: auto;
}

h2 {
  text-decoration-line: underline;
  text-decoration-thickness: 5px;
}

h3 {
  text-decoration-line: underline;
  text-decoration-thickness: 25%;
}

p {
  text-decoration-line: underline;
  text-decoration-color: red;
  text-decoration-style: double;
  text-decoration-thickness: 5px;
}
```

The Shorthand Property

The `text-decoration` property is a shorthand property for:

- `text-decoration-line` (required)
- `text-decoration-color` (optional)
- `text-decoration-style` (optional)
- `text-decoration-thickness` (optional)

Example

```
h1 {
  text-decoration: underline;
}

h2 {
  text-decoration: underline red;
}

h3 {
  text-decoration: underline red double;
}

p {
  text-decoration: underline red double 5px;
}
```

A Small Tip

All links in HTML are underlined by default. Sometimes you see that links are styled with no underline. The `text-decoration: none;` is used to remove the underline from links, like this:

Example

```
a {
  text-decoration: none;
}
```

All CSS text-decoration Properties

Property	Description
text-decoration	Sets all the text-decoration properties in one declaration
text-decoration-color	Specifies the color of the text-decoration
text-decoration-line	Specifies the kind of text decoration to be used (underline, overline, etc.)
text-decoration-style	Specifies the style of the text decoration (solid, dotted, etc.)
text-decoration-thickness	Specifies the thickness of the text decoration line

CSS Text Transformation

Text Transformation

The `text-transform` property is used to specify uppercase and lowercase letters in a text.

It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word:

Example

```
p.uppercase {
  text-transform: uppercase;
}

p.lowercase {
  text-transform: lowercase;
}

p.capitalize {
  text-transform: capitalize;
}
```

The CSS Text Transformation Property

Property	Description
text-transform	Controls the capitalization of text

CSS Text Spacing

Text Spacing

In this chapter you will learn about the following properties:

- text-indent
- letter-spacing
- line-height
- word-spacing
- white-space

Text Indentation

The text-indent property is used to specify the indentation of the first line of a text:

Example

```
p {
  text-indent: 50px;
}
```

Letter Spacing

The `letter-spacing` property is used to specify the space between the characters in a text.

The following example demonstrates how to increase or decrease the space between characters:

Example

```
h1 {
  letter-spacing: 5px;
}

h2 {
  letter-spacing: -2px;
}
```

Line Height

The `line-height` property is used to specify the space between lines:

Example

```
p.small {
  line-height: 0.8;
}

p.big {
  line-height: 1.8;
}
```

Word Spacing

The `word-spacing` property is used to specify the space between the words in a text.

The following example demonstrates how to increase or decrease the space between words:

Example

```
p.one {
  word-spacing: 10px;
}

p.two {
  word-spacing: -2px;
}
```

White Space

The `white-space` property specifies how white-space inside an element is handled.

This example demonstrates how to disable text wrapping inside an element:

Example

```
p {
  white-space: nowrap;
}
```

The CSS Text Spacing Properties

Property	Description
letter-spacing	Specifies the space between characters in a text
line-height	Specifies the line height
text-indent	Specifies the indentation of the first line in a text-block

white-space	Specifies how to handle white-space inside an element
word-spacing	Specifies the space between words in a text

CSS Text Shadow

Text Shadow

The `text-shadow` property adds shadow to text.

In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

Text shadow effect!

```
h1 {
  text-shadow: 2px 2px;
}
```

Next, add a color (red) to the shadow:

Text shadow effect!

```
h1 {
  text-shadow: 2px 2px red;
}
```

Then, add a blur effect (5px) to the shadow:

Text shadow effect!

```
h1 {
  text-shadow: 2px 2px 5px red;
}
```

More Text Shadow Examples

Text-shadow on a white text:

Example 1

```
h1 {
  color: white;
  text-shadow: 2px 2px 4px #000000;
}
```

Example 2

Text-shadow with red neon glow:

```
h1 {
  text-shadow: 0 0 3px #ff0000;
}
```

Example 3

Text-shadow with red and blue neon glow:

```
h1 {
  text-shadow: 0 0 3px #ff0000, 0 0 5px #0000ff;
}
```

Example 4

```
h1 {
  color: white;
  text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
}
```

Exercise:

Change the text color of all <p> elements to "red".

```
<style>
p {
  : red;
}
</style>

<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph</p>
  <p>This is a paragraph</p>
```

`</body>`

CSS Fonts

Choosing the right font for your website is important!

Font Selection is Important

Choosing the right font has a huge impact on how the readers experience a website.

The right font can create a strong identity for your brand.

Using a font that is easy to read is important. The font adds value to your text. It is also important to choose the correct color and text size for the font.

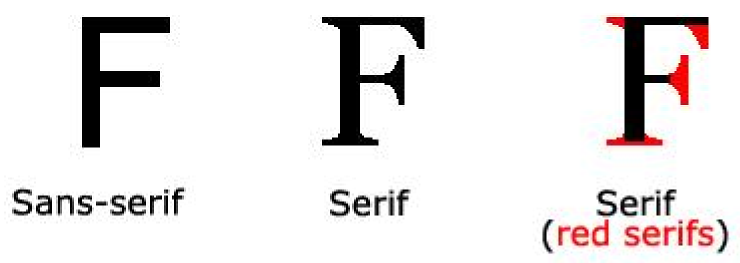
Generic Font Families

In CSS there are five generic font families:

1. **Serif** fonts have a small stroke at the edges of each letter. They create a sense of formality and elegance.
2. **Sans-serif** fonts have clean lines (no small strokes attached). They create a modern and minimalistic look.
3. **Monospace** fonts - here all the letters have the same fixed width. They create a mechanical look.
4. **Cursive** fonts imitate human handwriting.
5. **Fantasy** fonts are decorative/playful fonts.

All the different font names belong to one of the generic font families.

Difference Between Serif and Sans-serif Fonts



Note: On computer screens, sans-serif fonts are considered easier to read than serif fonts.

Some Font Examples

Generic Font Family	Examples of Font Names
Serif	Times New Roman Georgia Garamond
Sans-serif	Arial Verdana Helvetica
Monospace	Courier New Lucida Console Monaco
Cursive	Brush Script MT Lucida Handwriting
Fantasy	Papyrus

The CSS font-family Property

In CSS, we use the `font-family` property to specify the font of a text.

Note: If the font name is more than one word, it must be in quotation marks, like: "Times New Roman".

Tip: The `font-family` property should hold several font names as a "fallback" system, to ensure maximum compatibility between browsers/operating systems. Start with the font you want, and end with a generic family (to let the browser pick a similar font in the generic family, if no other fonts are available). The font names should be separated with comma. Read more about fallback fonts in the next chapter.

Example

Specify some different fonts for three paragraphs:

```
.p1 {
  font-family: "Times New Roman", Times, serif;
}

.p2 {
  font-family: Arial, Helvetica, sans-serif;
}

.p3 {
  font-family: "Lucida Console", "Courier New", monospace;
}
```

CSS Web Safe Fonts

What are Web Safe Fonts?

Web safe fonts are fonts that are universally installed across all browsers and devices.

Fallback Fonts

However, there are no 100% completely web safe fonts. There is always a chance that a font is not found or is not installed properly.

Therefore, it is very important to always use fallback fonts.

This means that you should add a list of similar "backup fonts" in the `font-family` property. If the first font does not work, the browser will try the next one, and the next one, and so on. Always end the list with a generic font family name.

Example

Here, there are three font types: Tahoma, Verdana, and sans-serif. The second and third fonts are backups, in case the first one is not found.

```
p {
font-family: Tahoma, Verdana, sans-serif;
}
```

Best Web Safe Fonts for HTML and CSS

The following list are the best web safe fonts for HTML and CSS:

- Arial (sans-serif)
- Verdana (sans-serif)
- Tahoma (sans-serif)
- Trebuchet MS (sans-serif)
- Times New Roman (serif)
- Georgia (serif)
- Garamond (serif)
- Courier New (monospace)
- Brush Script MT (cursive)

Note: Before you publish your website, always check how your fonts appear on different browsers and devices, and always use fallback fonts!

Arial (sans-serif)

Arial is the most widely used font for both online and printed media. Arial is also the default font in Google Docs.

Arial is one of the safest web fonts, and it is available on all major operating systems.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Verdana (sans-serif)

Verdana is a very popular font. Verdana is easily readable even for small font sizes.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Tahoma (sans-serif)

The Tahoma font has less space between the characters.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Trebuchet MS (sans-serif)

Trebuchet MS was designed by Microsoft in 1996. Use this font carefully. Not supported by all mobile operating systems.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Times New Roman (serif)

Times New Roman is one of the most recognizable fonts in the world. It looks professional and is used in many newspapers and "news" websites. It is also the primary font for Windows devices and applications.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Georgia (serif)

Georgia is an elegant serif font. It is very readable at different font sizes, so it is a good candidate for mobile-responsive design.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Garamond (serif)

Garamond is a classical font used for many printed books. It has a timeless look and good readability.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Courier New (monospace)

Courier New is the most widely used monospace serif font. Courier New is often used with coding displays, and many email providers use it as their default font. Courier New is also the standard font for movie screenplays.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Brush Script MT (cursive)

The Brush Script MT font was designed to mimic handwriting. It is elegant and sophisticated, but can be hard to read. Use it carefully.

Example

Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet.

0 1 2 3 4 5 6 7 8 9

Tip: Also check out all available Google Fonts and how to use them.

CSS Font Fallbacks

Commonly Used Font Fallbacks

Below are some commonly used font fallbacks, organized by the 5 generic font families:

- **Serif**
- **Sans-serif**
- **Monospace**
- **Cursive**
- **Fantasy**

Serif Fonts

font-family	Example text
"Times New Roman", Times, serif	**This is a Heading** This is a paragraph.
Georgia, serif	**This is a Heading** This is a paragraph.
Garamond, serif	**This is a Heading** This is a paragraph.

Sans-Serif Fonts

font-family	Example text
Arial, Helvetica, sans-serif	**This is a Heading** This is a paragraph.
Tahoma, Verdana, sans-serif	**This is a Heading** This is a paragraph.

"Trebuchet MS", Helvetica, sans-serif	**This is a Heading** This is a paragraph.
Geneva, Verdana, sans-serif	**This is a Heading** This is a paragraph.

Monospace Fonts

font-family	Example text
"Courier New", Courier, monospace	**This is a Heading** This is a paragraph.

Cursive Fonts

font-family	Example text
"Brush Script MT", cursive	**This is a Heading** This is a paragraph.

Fantasy Fonts

font-family	Example text
Copperplate, Papyrus, fantasy	This is a Heading This is a paragraph.

Tip: Also check out all available Google Fonts and how to use them.

CSS Font Style

Font Style

The `font-style` property is mostly used to specify italic text.

This property has three values:

- normal - The text is shown normally
- italic - The text is shown in italics
- oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

Example

```
p.normal {
  font-style: normal;
}

p.italic {
  font-style: italic;
}
```

```
p.oblique {
  font-style: oblique;
}
```

Font Weight

The `font-weight` property specifies the weight of a font:

Example

```
p.normal {
  font-weight: normal;
}

p.thick {
  font-weight: bold;
}
```

Font Variant

The `font-variant` property specifies whether or not a text should be displayed in a small-caps font.

In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.

Example

```
p.normal {
  font-variant: normal;
}

p.small {
  font-variant: small-caps;
}
```

CSS Font Size

Font Size

The `font-size` property sets the size of the text.

Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs.

Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs.

The font-size value can be an absolute, or relative size.

Absolute size:

- Sets the text to a specified size
- Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
- Absolute size is useful when the physical size of the output is known

Relative size:

- Sets the size relative to surrounding elements
- Allows a user to change the text size in browsers

Note: If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em).

Set Font Size With Pixels

Setting the text size with pixels gives you full control over the text size:

Example

```
h1 {
  font-size: 40px;
}

h2 {
  font-size: 30px;
}
```

```
p {
  font-size: 14px;
}
```

Tip: If you use pixels, you can still use the zoom tool to resize the entire page.

Set Font Size With Em

To allow users to resize the text (in the browser menu), many developers use em instead of pixels.

1em is equal to the current font size. The default text size in browsers is 16px. So, the default size of 1em is 16px.

The size can be calculated from pixels to em using this formula: *pixels*/16=*em*

Example

```
h1 {
  font-size: 2.5em; /* 40px/16=2.5em */
}

h2 {
  font-size: 1.875em; /* 30px/16=1.875em */
}

p {
  font-size: 0.875em; /* 14px/16=0.875em */
}
```

In the example above, the text size in em is the same as the previous example in pixels. However, with the em size, it is possible to adjust the text size in all browsers.

Unfortunately, there is still a problem with older versions of Internet Explorer. The text becomes larger than it should when made larger, and smaller than it should when made smaller.

Use a Combination of Percent and Em

The solution that works in all browsers, is to set a default font-size in percent for the <body> element:

Example

```
body {
  font-size: 100%;
}

h1 {
  font-size: 2.5em;
}

h2 {
  font-size: 1.875em;
}

p {
  font-size: 0.875em;
}
```

Our code now works great! It shows the same text size in all browsers, and allows all browsers to zoom or resize the text!

Responsive Font Size

The text size can be set with a `vw` unit, which means the "viewport width".

That way the text size will follow the size of the browser window:

```
<h1 style="font-size:10vw">Hello World</h1>
```

Resize the browser window to see how the font size scales.

Viewport is the browser window size. 1vw = 1% of viewport width. If the viewport is 50cm wide, 1vw is 0.5cm.

CSS Google Fonts

Google Fonts

If you do not want to use any of the standard fonts in HTML, you can use Google Fonts.

Google Fonts are free to use, and have more than 1000 fonts to choose from.

How To Use Google Fonts

Just add a special style sheet link in the <head> section and then refer to the font in the CSS.

Example

Here, we want to use a font named "Sofia" from Google Fonts:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=So
fia">
<style>
body {
  font-family: "Sofia", sans-serif;
}
</style>
</head>
```

Example

Here, we want to use a font named "Trirong" from Google Fonts:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?famil
y=Trirong">
```

```
<style>
body {
  font-family: "Trirong", serif;
}
</style>
</head>
```

Example

Here, we want to use a font named "Audiowide" from Google Fonts:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?famil
y=Audiowide">
<style>
body {
  font-family: "Audiowide", sans-serif;
}
</style>
</head>
```

Note: Requesting multiple fonts may slow down your web pages! So be careful about that.

Styling Google Fonts

Of course you can style Google Fonts as you like, with CSS!

Example

Style the "Sofia" font:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=So
fia">
<style>
body {
  font-family: "Sofia", sans-serif;
  font-size: 30px;
  text-shadow: 3px 3px 3px #ababab;
}
</style>
</head>
```

Enabling Font Effects

Google has also enabled different font effects that you can use.

First add `effect=effectname` to the Google API, then add a special class name to the element that is going to use the special effect. The class name always starts with `font-effect-` and ends with the `effectname`.

Example

Add the fire effect to the "Sofia" font:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=So
fia&effect=fire">
<style>
body {
  font-family: "Sofia", sans-serif;
  font-size: 30px;
}
</style>
</head>
<body>

<h1 class="font-effect-fire">Sofia on Fire</h1>

</body>
```

To request multiple font effects, just separate the effect names with a pipe character (`|`), like this:

Example

Add multiple effects to the "Sofia" font:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=So
fia&effect=neon|outline|emboss|shadow-multiple">
<style>
```

```
body {
  font-family: "Sofia", sans-serif;
  font-size: 30px;
}
</style>
</head>
<body>

<h1 class="font-effect-neon">Neon Effect</h1>
<h1 class="font-effect-outline">Outline Effect</h1>
<h1 class="font-effect-emboss">Emboss Effect</h1>
<h1 class="font-effect-shadow-multiple">Multiple Shadow Effect</h1>

</body>
```

To request multiple font effects, just separate the effect names with a pipe character (|), like this:

Example

Add multiple effects to the "Sofia" font:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=So
fia&effect=neon|outline|emboss|shadow-multiple">
<style>
body {
  font-family: "Sofia", sans-serif;
  font-size: 30px;
}
</style>
</head>
<body>

<h1 class="font-effect-neon">Neon Effect</h1>
<h1 class="font-effect-outline">Outline Effect</h1>
<h1 class="font-effect-emboss">Emboss Effect</h1>
<h1 class="font-effect-shadow-multiple">Multiple Shadow Effect</h1>

</body>
```

CSS Great Font Pairings

Great font pairings are essential to great design.

Font Pairing Rules

Here are some basic rules to create great font pairings:

1. Complement

It is always safe to find font pairings that complement one another.

A great font combination should harmonize, without being too similar or too different.

2. Use Font Superfamilies

A font superfamily is a set of fonts designed to work well together. So, using different fonts within the same superfamily is safe.

For example, the Lucida superfamily contains the following fonts: Lucida Sans, Lucida Serif, Lucida Typewriter Sans, Lucida Typewriter Serif and Lucida Math.

3. Contrast is King

Two fonts that are too similar will often conflict. However, contrasts, done the right way, brings out the best in each font.

Example: Combining serif with sans serif is a well known combination.

A strong superfamily includes both serif and sans serif variations of the same font (e.g. Lucida and Lucida Sans).

4. Choose Only One Boss

One font should be the boss. This establishes a hierarchy for the fonts on your page. This can be achieved by varying the size, weight and color.

Example

No doubt "Georgia" is the boss here:

```
body {
  background-color: black;
  font-family: Verdana, sans-serif;
  font-size: 16px;
  color: gray;
}

h1 {
  font-family: Georgia, serif;
  font-size: 60px;
  color: white;
}
```

Below, we have shown some popular font pairings that will suit many brands and contexts.

Georgia and Verdana

Georgia and Verdana is a classic combination. It also sticks to the web safe font standards:

Example

Use the "Georgia" font for headings, and "Verdana" for text:

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

```
<style>
body {
  background-color: black;
  font-family: Verdana, sans-serif;
  font-size: 16px;
  color: gray;
}

h1 {
  font-family: Georgia, serif;
  font-size: 60px;
  color: white;
}
</style>
</head>
<body>

<h1>Beautiful Norway</h1>

<p>Norway has a total area of 385,252 square kilometers and a population of 5,438,657
(December 2020). Norway is bordered by Sweden, Finland and Russia to the north-east, and the
Skagerrak to the south, with Denmark on the other side.</p>

<p>Norway has beautiful mountains, glaciers and stunning fjords. Oslo, the capital, is a city of
green spaces and museums. Bergen, with colorful wooden houses, is the starting point for cruises
to the dramatic Sognefjord. Norway is also known for fishing, hiking and skiing.</p>

</body>
</html>
```

Beautiful Norway

Norway has a total area of 385,252 square kilometers and a population of 5,438,657 (December 2020). Norway is bordered by Sweden, Finland and Russia to the north-east, and the Skagerrak to the south, with Denmark on the other side.

Norway has beautiful mountains, glaciers and stunning fjords. Oslo, the capital, is a city of green spaces and museums. Bergen, with colorful wooden houses, is the starting point for cruises to the dramatic Sognefjord. Norway is also known for fishing, hiking and skiing.

CSS Font Property

The CSS Font Property

To shorten the code, it is also possible to specify all the individual font properties in one property.

The `font` property is a shorthand property for:

- `font-style`
- `font-variant`
- `font-weight`
- `font-size/line-height`
- `font-family`

Note: The `font-size` and `font-family` values are required. If one of the other values is missing, their default value are used.

Example

Use `font` to set several font properties in one declaration:

```
p.a {
  font: 20px Arial, sans-serif;
}

p.b {
  font: italic small-caps bold 12px/30px Georgia, serif;
}
```

Exercise:

Set the font for <h1> to "Verdana".

```
<style>
h1 {
  : Verdana;
}
</style>

<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph</p>
</body>
```

CSS Pseudo-classes

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

Syntax

The syntax of pseudo-classes:

```
selector:pseudo-class {
  property: value;
}
```

Anchor Pseudo-classes

Links can be displayed in different ways:

```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}

/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

Note: `a:hover` MUST come after `a:link` and `a:visited` in the CSS definition in order to be effective! `a:active` MUST come after `a:hover` in the CSS definition in order to be effective! Pseudo-class names are not case-sensitive.

Pseudo-classes and HTML Classes

Pseudo-classes can be combined with HTML classes:

When you hover over the link in the example, it will change color:

Example

```
a.highlight:hover {
  color: #ff0000;
}
```

Hover on <div>

An example of using the `:hover` pseudo-class on a <div> element:

Example

```
div:hover {
  background-color: blue;
}
```

Simple Tooltip Hover

Hover over a <div> element to show a <p> element (like a tooltip):

Hover over me to show the <p> element.

```
p {
  display: none;
  background-color: yellow;
  padding: 20px;
}

div:hover p {
  display: block;
}
```

CSS - The :first-child Pseudo-class

The `:first-child` pseudo-class matches a specified element that is the first child of another element.

Match the first <p> element

In the following example, the selector matches any <p> element that is the first child of any element:

Example

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

Match the first <i> element in all <p> elements

In the following example, the selector matches the first <i> element in all <p> elements:

Example

```
p i:first-child {
  color: blue;
}
```

Match all <i> elements in all first child <p> elements

In the following example, the selector matches all <i> elements in <p> elements that are the first child of another element:

Example

```
p:first-child i {
  color: blue;
}
```

CSS - The :lang Pseudo-class

The `:lang` pseudo-class allows you to define special rules for different languages.

In the example below, `:lang` defines the quotation marks for <q> elements with lang="no":

Example

```
<html>
<head>
<style>
q:lang(no) {
  quotes: "~" "~";
}
</style>
</head>
<body>

<p>Some text <q lang="no">A quote in a paragraph</q> Some text.</p>

</body>
</html>
```

Exercise:

Set the background-color to red, when you mouse over a link.

```
<style> {
  background-color: red;
}
</style>

<body>

<h1>This is a header.</h1>
<p>This is a paragraph.</p>
<a href="https://w3schools.com">This is a link.</a>

</body>
```

All CSS Pseudo Classes

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element

:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a specified range
:invalid	input:invalid	Selects all <input> elements with an invalid value
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"
:last-child	p:last-child	Selects every <p> elements that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> element
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent

:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a specified range
:read-only	input:read-only	Selects <input> elements with a "readonly" attribute specified
:read-write	input:read-write	Selects <input> elements with no "readonly" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified
:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

All CSS Pseudo Elements

Selector	Example	Example description
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::marker	::marker	Selects the markers of list items
::selection	p::selection	Selects the portion of an element that is selected by a user

CSS Pseudo-elements

What are Pseudo-Elements?

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

Syntax

The syntax of pseudo-elements:

```
selector::pseudo-element {
    property:value;
}
```

Notice the double colon notation - ::first-line versus :first-line

The double colon replaced the single-colon notation for pseudo-elements in CSS3. This was an attempt from W3C to distinguish between **pseudo-classes** and **pseudo-elements**.

The single-colon syntax was used for both pseudo-classes and pseudo-elements in CSS2 and CSS1.

For backward compatibility, the single-colon syntax is acceptable for CSS2 and CSS1 pseudo-elements.

The ::first-line Pseudo-element

The ::first-line pseudo-element is used to add a special style to the first line of a text.

The following example formats the first line of the text in all <p> elements:

Example

```
p::first-line {
    color: #ff0000;
    font-variant: small-caps;
}
```

Note: The ::first-line pseudo-element can only be applied to block-level elements.

The following properties apply to the ::first-line pseudo-element:

- font properties
- color properties
- background properties
- word-spacing
- letter-spacing
- text-decoration
- vertical-align
- text-transform
- line-height
- clear

The ::first-letter Pseudo-element

The ::first-letter pseudo-element is used to add a special style to the first letter of a text.

The following example formats the first letter of the text in all <p> elements:

Example

```
p::first-letter {
    color: #ff0000;
    font-size: xx-large;
}
```

Note: The ::first-letter pseudo-element can only be applied to block-level elements.

The following properties apply to the ::first-letter pseudo- element:

- font properties
- color properties
- background properties
- margin properties
- padding properties
- border properties
- text-decoration
- vertical-align (only if "float" is "none")
- text-transform
- line-height
- float
- clear

Pseudo-elements and CSS Classes

Pseudo-elements can be combined with CSS classes:

Example

```
p.intro::first-letter {
    color: #ff0000;
    font-size:200%;
}
```

The example above will display the first letter of paragraphs with class="intro", in red and in a larger size.

Multiple Pseudo-elements

Several pseudo-elements can also be combined.

In the following example, the first letter of a paragraph will be red, in an xx-large font size. The rest of the first line will be blue, and in small-caps. The rest of the paragraph will be the default font size and color:

Example

```
p::first-letter {
    color: #ff0000;
    font-size: xx-large;
}

p::first-line {
    color: #0000ff;
    font-variant: small-caps;
}
```

CSS - The ::before Pseudo-element

The ::before pseudo-element can be used to insert some content before the content of an element.

The following example inserts an image before the content of each <h1> element:

Example

```
h1::before {
    content: url(smiley.gif);
}
```

CSS - The ::after Pseudo-element

The ::after pseudo-element can be used to insert some content after the content of an element.

The following example inserts an image after the content of each <h1> element:

Example

```
h1::after {
    content: url(smiley.gif);
}
```

CSS - The ::selection Pseudo-element

The ::selection pseudo-element matches the portion of an element that is selected by a user.

The following CSS properties can be applied to ::selection: color, background, cursor, and outline.

The following example makes the selected text red on a yellow background:

Example

```
::selection {
    color: red;
    background: yellow;
}
```

Week 6:

HTML Canvas Graphics

What is HTML Canvas?

The HTML `<canvas>` element is used to draw graphics, on the fly, via JavaScript.

The `<canvas>` element is only a container for graphics. You must use JavaScript to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

Canvas Examples

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Note: Always specify an `id` attribute (to be referred to in a script), and a `width` and `height` attribute to define the size of the canvas. To add a border, use the `style` attribute.

Add a JavaScript

After creating the rectangular canvas area, you must add a JavaScript to do the drawing.

Here are some examples:

Draw a Line

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

Draw a Circle

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

Draw a Text

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Hello World", 10, 50);
</script>
```

Stroke Text

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.strokeText("Hello World", 10, 50);
</script>
```

Draw Linear Gradient

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create gradient
var grd = ctx.createLinearGradient(0, 0, 200, 0);
grd.addColorStop(0, "red");
grd.addColorStop(1, "white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10, 10, 150, 80);
</script>
```

Draw Circular Gradient

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create gradient
var grd = ctx.createRadialGradient(75, 50, 5, 90, 60, 100);
```

```
grd.addColorStop(0, "red");
grd.addColorStop(1, "white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10, 10, 150, 80);
</script>
```

Draw Image

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var img = document.getElementById("scream");
ctx.drawImage(img, 10, 10);
</script>
```

HTML SVG Graphics

SVG defines vector-based graphics in XML format.

What is SVG?

- SVG stands for Scalable Vector Graphics
- SVG is used to define graphics for the Web
- SVG is a W3C recommendation

The HTML <svg> Element

The HTML `<svg>` element is a container for SVG graphics.

SVG has several methods for drawing paths, boxes, circles, text, and graphic images.

SVG Circle

Example

```
<!DOCTYPE html>
<html>
<body>
```

```
<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-
width="4" fill="yellow" />
</svg>

</body>
</html>
```

SVG Rectangle

Example

```
<svg width="400" height="100">
  <rect width="400" height="100" style="fill:rgb(0,0,255);stroke-
width:10;stroke:rgb(0,0,0)" />
</svg>
```

SVG Rounded Rectangle

```
<svg width="400" height="180">
  <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
  style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />
</svg>
```

SVG Star

Example

```
<svg width="300" height="200">
  <polygon points="100,10 40,198 190,78 10,78 160,198"
  style="fill:lime;stroke:purple;stroke-width:5;fill-
rule:evenodd;" />
</svg>
```

SVG Logo

Example

```
<svg height="130" width="500">
  <defs>
    <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-
```

```
opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-
opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="100" cy="70" rx="85" ry="55" fill="url(#grad1)" />
  <text fill="#ffffff" font-size="45" font-
family="Verdana" x="50" y="86">SVG</text>
  Sorry, your browser does not support inline SVG.
</svg>
```

Differences Between SVG and Canvas

SVG is a language for describing 2D graphics in XML.

Canvas draws 2D graphics, on the fly (with JavaScript).

SVG is XML based, which means that every element is available within the SVG DOM. You can attach JavaScript event handlers for an element.

In SVG, each drawn shape is remembered as an object. If attributes of an SVG object are changed, the browser can automatically re-render the shape.

Canvas is rendered pixel by pixel. In canvas, once the graphic is drawn, it is forgotten by the browser. If its position should be changed, the entire scene needs to be redrawn, including any objects that might have been covered by the graphic.

Comparison of Canvas and SVG

The table below shows some important differences between Canvas and SVG:

Canvas	SVG
• Resolution dependent • No support for event handlers • Poor text rendering capabilities • You can save the resulting image as .png or .jpg • Well suited for graphic-	• Resolution independent • Support for event handlers • Best suited for applications with large rendering areas (Google Maps) • Slow rendering if complex (anything that uses the DOM a lot will be slow) • Not suited for game applications

intensive games

HTML Multimedia(Audio and Video)

Multimedia on the web is sound, music, videos, movies, and animations.

What is Multimedia?

Multimedia comes in many different formats. It can be almost anything you can hear or see, like images, music, sound, videos, records, films, animations, and more.

Web pages often contain multimedia elements of different types and formats.

Browser Support

The first web browsers had support for text only, limited to a single font in a single color.

Later came browsers with support for colors, fonts, images, and multimedia!

Multimedia Formats

Multimedia elements (like audio or video) are stored in media files.

The most common way to discover the type of a file, is to look at the file extension.

Multimedia files have formats and different extensions like: .wav, .mp3, .mp4, .mpg, .wmv, and .avi.

Common Video Formats

There are many video formats out there.

The MP4, WebM, and Ogg formats are supported by HTML.

The MP4 format is recommended by YouTube.

Format	File	Description
MPEG	.mpg .mpeg	MPEG. Developed by the Moving Pictures Expert Group. The first popular video format on the web. Not supported anymore in HTML.
AVI	.avi	AVI (Audio Video Interleave). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
WMV	.wmv	WMV (Windows Media Video). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
QuickTime	.mov	QuickTime. Developed by Apple. Commonly used in video cameras and TV hardware. Plays well on Apple computers, but not in web browsers.
RealVideo	.rm .ram	RealVideo. Developed by Real Media to allow video streaming with low bandwidths. Does not play in web browsers.
Flash	.swf .flv	Flash. Developed by Macromedia. Often requires an extra component (plug-in) to play in web browsers.
Ogg	.ogg	Theora Ogg. Developed by the Xiph.Org Foundation. Supported by HTML.
WebM	.webm	WebM. Developed by Mozilla, Opera, Adobe, and Google. Supported by HTML.
MPEG-4 or MP4	.mp4	MP4. Developed by the Moving Pictures Expert Group. Commonly used in video cameras and TV hardware. Supported by all browsers and recommended by

		YouTube.

Note: Only MP4, WebM, and Ogg video are supported by the HTML standard.

Common Audio Formats

MP3 is the best format for compressed recorded music. The term MP3 has become synonymous with digital music.

If your website is about recorded music, MP3 is the choice.

Format	File	Description
MIDI	.mid .midi	MIDI (Musical Instrument Digital Interface). Main format for all electronic music devices like synthesizers and PC sound cards. MIDI files do not contain sound, but digital notes that can be played by electronics. Plays well on all computers and music hardware, but not in web browsers.
RealAudio	.rm .ram	RealAudio. Developed by Real Media to allow streaming of audio with low bandwidths. Does not play in web browsers.
WMA	.wma	WMA (Windows Media Audio). Developed by Microsoft. Plays well on Windows computers, but not in web browsers.
AAC	.aac	AAC (Advanced Audio Coding). Developed by Apple as the default format for iTunes. Plays well on Apple computers, but not in web browsers.
WAV	.wav	WAV. Developed by IBM and Microsoft. Plays well on Windows, Macintosh, and Linux operating systems. Supported by HTML.
Ogg	.ogg	Ogg. Developed by the Xiph.Org Foundation. Supported by HTML.
MP3	.mp3	MP3 files are actually the sound part of MPEG files. MP3 is the most popular format for music players. Combines good compression (small files) with high quality.

		Supported by all browsers.
MP4	.mp4	MP4 is a video format, but can also be used for audio. Supported by all browsers.

Note: Only MP3, WAV, and Ogg audio are supported by the HTML standard.

HTML Video

```
<!DOCTYPE html>
<html>
<body>

<video width="400" controls>
  <source src="mov_bbb.mp4" type="video/mp4">
  <source src="mov_bbb.ogg" type="video/ogg">
  Your browser does not support HTML video.
</video>

<p>
Video courtesy of
<a href="https://www.bigbuckbunny.org/" target="_blank">Big Buck Bunny</a>.
</p>

</body>
</html>
```

The HTML <video> Element

To show a video in HTML, use the `<video>` element:

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

How it Works

The `controls` attribute adds video controls, like play, pause, and volume.

It is a good idea to always include `width` and `height` attributes. If height and width are not set, the page might flicker while the video loads.

The `<source>` element allows you to specify alternative video files which the browser may choose from. The browser will use the first recognized format.

The text between the `<video>` and `</video>` tags will only be displayed in browsers that do not support the `<video>` element.

HTML <video> Autoplay

To start a video automatically, use the `autoplay` attribute:

Example

```
<video width="320" height="240" autoplay>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

Note: Chromium browsers do not allow autoplay in most cases. However, muted autoplay is always allowed.

Add `muted` after `autoplay` to let your video start playing automatically (but muted):

Example

```
<video width="320" height="240" autoplay muted>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

HTML Video Formats

There are three supported video formats: MP4, WebM, and Ogg. The browser support for the different formats is:

Browser	MP4	WebM	Ogg
Edge	YES	YES	YES
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

HTML Video - Media Types

File Format	Media Type
MP4	video/mp4
WebM	video/webm
Ogg	video/ogg

HTML Video - Methods, Properties, and Events

The HTML DOM defines methods, properties, and events for the `<video>` element.

This allows you to load, play, and pause videos, as well as setting duration and volume.

There are also DOM events that can notify you when a video begins to play, is paused, etc.

Example: Using JavaScript

```
<!DOCTYPE html>
<html>
<body>

<div style="text-align:center">
  <button onclick="playPause()">Play/Pause</button>
  <button onclick="makeBig()">Big</button>
  <button onclick="makeSmall()">Small</button>
  <button onclick="makeNormal()">Normal</button>
  <br><br>
  <video id="video1" width="420">
    <source src="mov_bbb.mp4" type="video/mp4">
    <source src="mov_bbb.ogg" type="video/ogg">
    Your browser does not support HTML video.
  </video>
</div>

<script>
var myVideo = document.getElementById("video1");

function playPause() {
  if (myVideo.paused)
    myVideo.play();
  else
    myVideo.pause();
}

function makeBig() {
    myVideo.width = 560;
}

function makeSmall() {
    myVideo.width = 320;
}
```

```
function makeNormal() {
   myVideo.width = 420;
}
</script>

<p>Video courtesy of <a href="https://www.bigbuckbunny.org/"
target="_blank">Big Buck Bunny</a>.</p>

</body>
</html>
```

HTML Video Tags

Tag	Description
<video>	Defines a video or movie
<source>	Defines multiple media resources for media elements, such as <video> and <audio>
<track>	Defines text tracks in media players

HTML Audio

The HTML `<audio>` element is used to play an audio file on a web page.

The HTML <audio> Element

To play an audio file in HTML, use the `<audio>` element:

Example

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
```

```
Your browser does not support the audio element.
</audio>
```

HTML Audio - How It Works

The `controls` attribute adds audio controls, like play, pause, and volume.

The `<source>` element allows you to specify alternative audio files which the browser may choose from. The browser will use the first recognized format.

The text between the `<audio>` and `</audio>` tags will only be displayed in browsers that do not support the `<audio>` element.

HTML <audio> Autoplay

To start an audio file automatically, use the `autoplay` attribute:

Example

```
<audio controls autoplay>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

Note: Chromium browsers do not allow autoplay in most cases. However, muted autoplay is always allowed.

Add `muted` after `autoplay` to let your audio file start playing automatically (but muted):

Example

```
<audio controls autoplay muted>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

HTML Audio Formats

There are three supported audio formats: MP3, WAV, and OGG. The browser support for the different formats is:

Browser	MP3	WAV	OGG
Edge/IE	YES	YES*	YES*
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

HTML Audio - Media Types

File Format	Media Type
MP3	audio/mpeg
OGG	audio/ogg
WAV	audio/wav

HTML Audio - Methods, Properties, and Events

The HTML DOM defines methods, properties, and events for the `<audio>` element.

This allows you to load, play, and pause audios, as well as set duration and volume.

There are also DOM events that can notify you when an audio begins to play, is paused, etc.

HTML Audio Tags

Tag	Description
<audio>	Defines sound content
<source>	Defines multiple media resources for media elements, such as <video> and <audio>

HTML YouTube Videos

The easiest way to play videos in HTML, is to use YouTube.

Struggling with Video Formats?

Converting videos to different formats can be difficult and time-consuming.

An easier solution is to let YouTube play the videos in your web page.

YouTube Video Id

YouTube will display an id (like tgbNymZ7vqY), when you save (or play) a video.

You can use this id, and refer to your video in the HTML code.

Playing a YouTube Video in HTML

To play your video on a web page, do the following:

- Upload the video to YouTube
- Take a note of the video id
- Define an `<iframe>` element in your web page
- Let the `src` attribute point to the video URL
- Use the `width` and `height` attributes to specify the dimension of the player
- Add any other parameters to the URL (see below)

Example

```
<iframe width="420" height="315"
src="https://www.youtube.com/embed/tgbNymZ7vqY">
</iframe>
```

YouTube Autoplay + Mute

You can let your video start playing automatically when a user visits the page, by adding `autoplay=1` to the YouTube URL. However, automatically starting a video is annoying for your visitors!

Note: Chromium browsers do not allow autoplay in most cases. However, muted autoplay is always allowed.

Add `mute=1` after `autoplay=1` to let your video start playing automatically (but muted).

YouTube - Autoplay + Muted

```
<iframe width="420" height="315"
src="https://www.youtube.com/embed/tgbNymZ7vqY?autoplay=1&mute=1">
</iframe>
```

YouTube Playlist

A comma separated list of videos to play (in addition to the original URL).

YouTube Loop

Add `loop=1` to let your video loop forever.

Value 0 (default): The video will play only once.

Value 1: The video will loop (forever).

YouTube - Loop

```
<iframe width="420" height="315"
src="https://www.youtube.com/embed/tgbNymZ7vqY?playlist=tgbNymZ7vqY&loo
p=1">
</iframe>
```

YouTube Controls

Add `controls=0` to not display controls in the video player.

Value 0: Player controls does not display.

Value 1 (default): Player controls display.

YouTube - Controls

```
<iframe width="420" height="315"
src="https://www.youtube.com/embed/tgbNymZ7vqY?controls=0">
</iframe>
```

CSS Transitions

CSS Transitions

CSS transitions allows you to change property values smoothly, over a given duration.

In this chapter you will learn about the following properties:

- `transition`
- `transition-delay`
- `transition-duration`
- `transition-property`
- `transition-timing-function`

How to Use CSS 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

Note: If the duration part is not specified, the transition will have no effect, because the default value is 0.

The following example shows a 100px * 100px red <div> element. The <div> element has also specified a transition effect for the width property, with a duration of 2 seconds:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}

div:hover {
  width: 300px;
}
</style>
</head>
<body>
```

```
<h1>The transition Property</h1>


<p>Hover over the div element below, to see the transition
effect:</p>
<div></div>


</body>
</html>
```

The transition effect will start when the specified CSS property (width) changes value.

Now, let us specify a new value for the width property when a user mouses over the <div> element:

Example

```
div:hover {
  width: 300px;
}
```

Notice that when the cursor mouses out of the element, it will gradually change back to its original style.

Change Several Property Values

The following example adds a transition effect for both the width and height property, with a duration of 2 seconds for the width and 4 seconds for the height:

```
div {
  transition: width 2s, height 4s;
}

div:hover {
  width: 300px;
  height: 300px;
}
```

Specify the Speed Curve of the Transition

The `transition-timing-function` property specifies the speed curve of the transition effect.

The transition-timing-function property can have the following values:

- `ease` - specifies a transition effect with a slow start, then fast, then end slowly (this is 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
- `cubic-bezier(n,n,n,n)` - lets you define your own values in a cubic-bezier function

The following example shows some of the different speed curves that can be used:

Example

```
#div1 {transition-timing-function: linear;}
#div2 {transition-timing-function: ease;}
#div3 {transition-timing-function: ease-in;}
#div4 {transition-timing-function: ease-out;}
#div5 {transition-timing-function: ease-in-out;}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}

#div1 {transition-timing-function: linear;}
#div2 {transition-timing-function: ease;}
#div3 {transition-timing-function: ease-in;}
#div4 {transition-timing-function: ease-out;}
#div5 {transition-timing-function: ease-in-out;}

div:hover {
  width: 300px;
}
</style>
</head>
<body>

<h1>The transition-timing-function Property</h1>

<p>Hover over the div elements below, to see the different speed
curves:</p>

<div id="div1">linear</div><br>
<div id="div2">ease</div><br>
<div id="div3">ease-in</div><br>
<div id="div4">ease-out</div><br>
<div id="div5">ease-in-out</div><br>

</body>
</html>
```

Delay the Transition Effect

The `transition-delay` property specifies a delay (in seconds) for the transition effect.

The following example has a 1 second delay before starting:

Example

```
div {
  transition-delay: 1s;
}
```

Transition + Transformation

The following example adds a transition effect to the transformation:

Example

```
div {
  transition: width 2s, height 2s, transform 2s;
}
```

More Transition Examples

The CSS transition properties can be specified one by one, like this:

Example

```
div {
  transition-property: width;
  transition-duration: 2s;
  transition-timing-function: linear;
  transition-delay: 1s;
}
```

or by using the shorthand property `transition`:

Example

```
div {
  transition: width 2s linear 1s;
}
```

Exercise:

Add a 2 second transition effect for width changes of the <div> element.

```
<style>
div {
  width: 100px;
  height: 100px;
  background: red;
  : ;
}

div:hover {
  width: 300px;
}
</style>

<body>
  <div>This is a div</div>
</body>
```

CSS Transition Properties

The following table lists all the CSS transition properties:

Property	Description
transition	A shorthand property for setting the four transition properties into a single property
transition-delay	Specifies a delay (in seconds) for the transition effect
transition-duration	Specifies how many seconds or milliseconds a transition effect takes to complete
transition-property	Specifies the name of the CSS property the transition

	effect is for
transition-timing-function	Specifies the speed curve of the transition effect

CSS Animations

CSS Animations

CSS allows animation of HTML elements without using JavaScript or Flash!

In this chapter you will learn about the following properties:

- `@keyframes`
- `animation-name`
- `animation-duration`
- `animation-delay`
- `animation-iteration-count`
- `animation-direction`
- `animation-timing-function`
- `animation-fill-mode`
- `animation`

What are CSS Animations?

An animation lets an element gradually change from one style to another.

You can change as many CSS properties you want, as many times as you want.

To use CSS animation, you must first specify some keyframes for the animation.

Keyframes hold what styles the element will have at certain times.

The @keyframes Rule

When you specify CSS styles inside the `@keyframes` rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

The following example binds the "example" animation to the <div> element. The animation will last for 4 seconds, and it will gradually change the background-color of the <div> element from "red" to "yellow":

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}

@keyframes example {
  from {background-color: red;}
  to {background-color: yellow;}
}
</style>
</head>
<body>

<h1>CSS Animation</h1>
```

```
<div></div>

<p><b>Note:</b> When an animation is finished, it goes back
to its original style.</p>

</body>
</html>
```

Note: The `animation-duration` property defines how long an animation should take to complete. If the `animation-duration` property is not specified, no animation will occur, because the default value is 0s (0 seconds).

In the example above we have specified when the style will change by using the keywords "from" and "to" (which represents 0% (start) and 100% (complete)).

It is also possible to use percent. By using percent, you can add as many style changes as you like.

The following example will change the background-color of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

```
/* The animation code */
@keyframes example {
  0%   {background-color: red;}
  25%  {background-color: yellow;}
  50%  {background-color: blue;}
  100% {background-color: green;}
}

/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

The following example will change both the background-color and the position of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

Example

```
/* The animation code */
@keyframes example {
  0%   {background-color:red; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px; top:0px;}
  50%  {background-color:blue; left:200px; top:200px;}
  75%  {background-color:green; left:0px; top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}

/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

Delay an Animation

The `animation-delay` property specifies a delay for the start of an animation.

The following example has a 2 seconds delay before starting the animation:

Example

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-delay: 2s;
}
```

Negative values are also allowed. If using negative values, the animation will start as if it had already been playing for N seconds.

In the following example, the animation will start as if it had already been playing for 2 seconds:

Example

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-delay: -2s;
}
```

Set How Many Times an Animation Should Run

The `animation-iteration-count` property specifies the number of times an animation should run.

The following example will run the animation 3 times before it stops:

Example

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-iteration-count: 3;
}
```

The following example uses the value "infinite" to make the animation continue for ever:

Example

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-iteration-count: infinite;
}
```

Run Animation in Reverse Direction or Alternate Cycles

The `animation-direction` property specifies whether an animation should be played forwards, backwards or in alternate cycles.

The animation-direction property can have the following values:

- `normal` - The animation is played as normal (forwards). This is default
- `reverse` - The animation is played in reverse direction (backwards)
- `alternate` - The animation is played forwards first, then backwards
- `alternate-reverse` - The animation is played backwards first, then forwards

The following example will run the animation in reverse direction (backwards):

Example

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-direction: reverse;
}
```

The following example uses the value "alternate" to make the animation run forwards first, then backwards:

Example

```
div {
  width: 100px;
  height: 100px;

position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-iteration-count: 2;
  animation-direction: alternate;
}
```

The following example uses the value "alternate-reverse" to make the animation run backwards first, then forwards:

Example

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-iteration-count: 2;
  animation-direction: alternate-reverse;
}
```

Specify the Speed Curve of the Animation

The `animation-timing-function` property specifies the speed curve of the animation.

The animation-timing-function property can have the following values:

- `ease` - Specifies an animation with a slow start, then fast, then end slowly (this is 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
- `cubic-bezier(n,n,n,n)` - Lets you define your own values in a cubic-bezier function

The following example shows some of the different speed curves that can be used:

Example

```
#div1 {animation-timing-function: linear;}
#div2 {animation-timing-function: ease;}
#div3 {animation-timing-function: ease-in;}
#div4 {animation-timing-function: ease-out;}
#div5 {animation-timing-function: ease-in-out;}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 50px;
  background-color: red;
  font-weight: bold;
  position: relative;
  animation: mymove 5s infinite;
}
```

```
#div1 {animation-timing-function: linear;}
#div2 {animation-timing-function: ease;}
#div3 {animation-timing-function: ease-in;}
#div4 {animation-timing-function: ease-out;}
#div5 {animation-timing-function: ease-in-out;}

@keyframes mymove {
  from {left: 0px;}
  to {left: 300px;}
}
</style>
</head>
<body>

<h1>CSS Animation</h1>

<p>The animation-timing-function property specifies the speed curve
of the animation. The following example shows some of the different
speed curves that can be used:</p>

<div id="div1">linear</div>
<div id="div2">ease</div>
<div id="div3">ease-in</div>
<div id="div4">ease-out</div>
<div id="div5">ease-in-out</div>

</body>
</html>
```

Specify the fill-mode For an Animation

CSS animations do not affect an element before the first keyframe is played or after the last keyframe is played. The animation-fill-mode property can override this behavior.

The `animation-fill-mode` property specifies a style for the target element when the animation is not playing (before it starts, after it ends, or both).

The animation-fill-mode property can have the following values:

- `none` - Default value. Animation will not apply any styles to the element before or after it is executing
- `forwards` - The element will retain the style values that is set by the last keyframe (depends on animation-direction and animation-iteration-count)
- `backwards` - The element will get the style values that is set by the first keyframe (depends on animation-direction), and retain this during the animation-delay period
- `both` - The animation will follow the rules for both forwards and backwards, extending the animation properties in both directions

The following example lets the <div> element retain the style values from the last keyframe when the animation ends:

Example

```
div {
  width: 100px;
  height: 100px;
  background: red;
  position: relative;
  animation-name: example;
  animation-duration: 3s;
  animation-fill-mode: forwards;
}
```

The following example lets the <div> element get the style values set by the first keyframe before the animation starts (during the animation-delay period):

Example

```
div {
  width: 100px;
  height: 100px;
  background: red;
  position: relative;
  animation-name: example;
  animation-duration: 3s;
  animation-delay: 2s;
  animation-fill-mode: backwards;
}
```

The following example lets the <div> element get the style values set by the first keyframe before the animation starts, and retain the style values from the last keyframe when the animation ends:

Example

```
div {
  width: 100px;
  height: 100px;
  background: red;
  position: relative;
  animation-name: example;
  animation-duration: 3s;
  animation-delay: 2s;
  animation-fill-mode: both;
}
```

Animation Shorthand Property

The example below uses six of the animation properties:

Example

```
div {
  animation-name: example;
  animation-duration: 5s;
  animation-timing-function: linear;
  animation-delay: 2s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}
```

The same animation effect as above can be achieved by using the shorthand `animation` property:

Example

```
div {
  animation: example 5s linear 2s infinite alternate;
}
```

Exercise:

Add a 2 second animation for the <div> element, which changes the color from red to blue. Call the animation "example".

```
<style>
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: ;
  : 2s;
}

@keyframes example {
  from {: red;}
  to {: blue;}
}
</style>

<body>
  <div>This is a div</div>
</body>
```

CSS Animation Properties

The following table lists the @keyframes rule and all the CSS animation properties:

Property	Description
@keyframes	Specifies the animation code
animation	A shorthand property for setting all the animation properties
animation-delay	Specifies a delay for the start of an animation
animation-direction	Specifies whether an animation should be played forwards, backwards or in alternate cycles
animation-duration	Specifies how long time an animation should take to complete one cycle
animation-fill-mode	Specifies a style for the element when the animation is not playing (before it starts, after it ends, or both)
animation-iteration-count	Specifies the number of times an animation should be played
animation-name	Specifies the name of the @keyframes animation
animation-play-state	Specifies whether the animation is running or paused
animation-timing-function	Specifies the speed curve of the animation

JavaScript Introduction

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web.

JavaScript is easy to learn.

You will learn JavaScript from basic to advanced.

```
<!DOCTYPE html>
<html>
<body>

<h2>My First JavaScript</h2>

<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>

<p id="demo"></p>

</body>
</html>
```

Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages

3. **JavaScript** to program the behavior of web pages

Learning Speed

In this lesson, the learning speed is your choice.

Everything is up to you.

If you are struggling, take a break, or re-read the material.

Always make sure you understand examples.

The only way to become a clever programmer is to: Practice. Practice. Practice. Code. Code. Code !

Commonly Asked Questions

- How do I get JavaScript?
- Where can I download JavaScript?
- Is JavaScript Free?

You don't have to get or download JavaScript.

JavaScript is already running in your browser on your computer, on your tablet, and on your smart-phone.

JavaScript is free to use for everyone.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

JavaScript accepts both double and single quotes: like so

```
document.getElementById('demo').innerHTML = 'Hello JavaScript';
```

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the `src` (source) attribute of an `<img>` tag:

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the light</button>

</body>
</html>
```

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";
```

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the `display` style:

```
document.getElementById("demo").style.display = "none";
```

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "block";
```

JavaScript Where To

The <script> Tag

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Example

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

Old JavaScript examples may use a type attribute: <script type="text/javascript">.

The type attribute is not required. JavaScript is the default scripting language in HTML.

JavaScript Functions and Events

A JavaScript `function` is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an **event** occurs, like when the user clicks a button.

You will learn much more about functions and events in later chapters.

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

JavaScript in <head>

In this example, a JavaScript `function` is placed in the `<head>` section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>
```

```
<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>



</body>
</html>
```

JavaScript in <body>

In this example, a JavaScript `function` is placed in the `<body>` section of an HTML page.

The function is invoked (called) when a button is clicked:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Demo JavaScript in Body</h2>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

External JavaScript

Scripts can also be placed in external files:

External file: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the `src` (source) attribute of a `<script>` tag:

Example

```
<script src="myScript.js"></script>
```

You can place an external script reference in `<head>` or `<body>` as you like.

The script will behave as if it was located exactly where the `<script>` tag is located.

External scripts cannot contain `<script>` tags.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path

This example uses a **full URL** to link to myScript.js:

Example

```
<script src="https://www.javascript.com/js/myScript.js"></script>
```

This example uses a **file path** to link to myScript.js:

Example

```
<script src="/js/myScript.js"></script>
```

This example uses no path to link to myScript.js:

Example

```
<script src="myScript.js"></script>
```

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with `//`.

Any text between `//` and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

Example

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";

// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

his example uses a single line comment at the end of each line to explain the code:

Example

```
let x = 5;      // Declare x, give it the value of 5
let y = x + 2;  // Declare y, give it the value of x + 2
```

Multi-line Comments

Multi-line comments start with `/*` and end with `*/`.

Any text between `/*` and `*/` will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

It is most common to use single line comments.
Block comments are often used for formal documentation.

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding `//` in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

Example

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

This example uses a comment block to prevent execution of multiple lines:

Example

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

JavaScript Variables

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

In this first example, `x`, `y`, and `z` are undeclared variables.

They are automatically declared when first used:

Example

```
x = 5;
y = 6;
z = x + y;
```

Note

It is considered good programming practice to always declare variables before use.

From the examples you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

Example using var

```
var x = 5;
var y = 6;
var z = x + y;
```

Note

The var keyword was used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

The var keyword should only be used in code written for older browsers.

Example using let

```
let x = 5;
let y = 6;
let z = x + y;
```

Example using const

```
const x = 5;
const y = 6;
const z = x + y;
```

Mixed Example

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

The two variables price1 and price2 are declared with the const keyword.

These are constant values and cannot be changed.

The variable total is declared with the let keyword.

The value total can be changed.

When to Use var, let, or const?

1. Always declare variables

2. Always use `const` if the value should not be changed

3. Always use `const` if the type should not be changed (Arrays and Objects)

4. Only use `let` if you can't use `const`

5. Only use `var` if you MUST support old browsers.

Just Like Algebra

Just like in algebra, variables hold values:

```
let x = 5;
let y = 6;
```

Just like in algebra, variables are used in expressions:

Just like in algebra, variables are used in expressions:

```
let z = x + y;
```

From the example above, you can guess that the total is calculated to be 11.

Note

Variables are containers for storing values.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with $ and _ (but we will not use it in this tutorial).
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note

JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (`=`) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

Note

The "equal to" operator is written like `==` in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the `var` or the `let` keyword:

```
var carName;
```

or:

```
let carName;
```

After the declaration, the variable has no value (technically it is `undefined`).

To **assign** a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

In the example below, we create a variable called `carName` and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

Example

```
<p id="demo"></p>

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Note

It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with `let` and separate the variables by **comma**:

Example

```
let person = "John Doe", carName = "Volvo", price = 200;
```

A declaration can span multiple lines:

Example

```
let person = "John Doe",
carName = "Volvo",
price = 200;
```

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value `undefined`.

The variable carName will have the value `undefined` after the execution of this statement:

Example

```
let carName;
```

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable declared with `var`, it will not lose its value.

The variable `carName` will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

Note

You cannot re-declare a variable declared with `let` or `const`.

This will not work:

```
let carName = "Volvo";
let carName;
```

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like `=` and `+`:

Example

```
let x = 5 + 2 + 3;
```

You can also add strings, but strings will be concatenated:

Example

```
let x = "John" + " " + "Doe";
```

Also try this:

Example

```
let x = "5" + 2 + 3;
```

Note

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

Example

```
let x = 2 + 3 + "5";
```

JavaScript Dollar Sign $

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

Example

```
let $ = "Hello World";
let $$$ = 2;
let $myMoney = 5;
```

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function `$` is used to select HTML elements. In jQuery `$("p");` means "select all p elements".

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

Example

```
let _lastName = "Johnson";
let _x = 2;
let _100 = 5;
```

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

Exercise:

Create a variable called `carName` and assign the value `Volvo` to it.

JavaScript Data Types

JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date

Examples

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

Note

A JavaScript variable can hold any type of data.

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
let x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + "Volvo";
```

Note

When adding a number and a string, JavaScript will treat the number as a string.

Example

```
let x = 16 + "Volvo";
```

Example

```
let x = "Volvo" + 16;
```

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

JavaScript:

```
let x = 16 + 4 + "Volvo";
```

Result:

```
20Volvo
```

JavaScript:

```
let x = "Volvo" + 16 + 4;
```

Result:

```
Volvo164
```

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example

```
let x;       // Now x is undefined
x = 5;       // Now x is a Number
x = "John";  // Now x is a String
```

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

```
// Using double quotes:
let carName1 = "Volvo XC60";

// Using single quotes:
let carName2 = 'Volvo XC60';
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
// Single quote inside double quotes:
let answer1 = "It's alright";

// Single quotes inside double quotes:
let answer2 = "He is called 'Johnny'";

// Double quotes inside single quotes:
let answer3 = 'He is called "Johnny"';
```

You will learn more about **strings** later in this tutorial.

JavaScript Numbers

All JavaScript numbers are stored as decimal numbers (floating point).

Numbers can be written with, or without decimals:

Example

```
// With decimals:
let x1 = 34.00;

// Without decimals:
let x2 = 34;
```

Exponential Notation

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example

```
let y = 123e5;    // 12300000
let z = 123e-5;   // 0.00123
```

Note

Most programming languages have many number types:

Whole numbers (integers):
byte (8-bit), short (16-bit), int (32-bit), long (64-bit)

Real numbers (floating-point):
float (32-bit), double (64-bit).

**Javascript numbers are always one type:
double (64-bit floating point).**

You will learn more about **numbers** later in this tutorial.

JavaScript BigInt

All JavaScript numbers are stored in a a 64-bit floating-point format.

JavaScript BigInt is a new datatype (ES2020) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

Example

```
let x = BigInt("123456789012345678901234567890");
```

You will learn more about **BigInt** later in this tutorial.

JavaScript Booleans

Booleans can only have two values: `true` or `false`.

Example

```
let x = 5;
let y = 5;
let z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Booleans are often used in conditional testing.

You will learn more about **booleans** later in this tutorial.

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called `cars`, containing three items (car names):

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about **arrays** later in this tutorial.

JavaScript Objects

JavaScript objects are written with curly braces `{}`.

Object properties are written as name:value pairs, separated by commas.

Example

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

You will learn more about **objects** later in this tutorial.

The typeof Operator

You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.

The `typeof` operator returns the type of a variable or an expression:

Example

```
typeof ""                  // Returns "string"
typeof "John"              // Returns "string"
typeof "John Doe"          // Returns "string"
```

Example

```
typeof 0                   // Returns "number"
typeof 314                 // Returns "number"
typeof 3.14                // Returns "number"
typeof (3)                 // Returns "number"
typeof (3 + 4)             // Returns "number"
```

You will learn more about **typeof** later in this lesson.

Undefined

In JavaScript, a variable without a value, has the value `undefined`. The type is also `undefined`.

Example

```
let car;    // Value is undefined, type is undefined
```

Any variable can be emptied, by setting the value to `undefined`. The type will also be `undefined`.

Example

```
car = undefined;    // Value is undefined, type is undefined
```

Empty Values

An empty value has nothing to do with `undefined`.

An empty string has both a legal value and a type.

Example

```
let car = "";    // The value is "", the typeof is "string"
```

Exercise:

Use comments to describe the correct data type of the following variables:

```
let length = 16;           //
let lastName = "Johnson";   //
const x = {
  firstName: "John",
  lastName: "Doe"
};
```

****Week 7-8: Advanced Topics****
Week 7:

HTML Iframes

An HTML iframe is used to display a web page within a web page.

HTML Iframe Syntax

The HTML `<iframe>` tag specifies an inline frame.

An inline frame is used to embed another document within the current HTML document.

```
<iframe src="url" title="description"></iframe>
```

Tip: It is a good practice to always include a `title` attribute for the `<iframe>`. This is used by screen readers to read out what the content of the iframe is.

Iframe - Set Height and Width

Use the `height` and `width` attributes to specify the size of the iframe.

The height and width are specified in pixels by default:

Example

```
<iframe src="demo_iframe.htm" height="200" width="300" title="Iframe
Example"></iframe>
```

Or you can add the `style` attribute and use the CSS `height` and `width` properties:

Example

```
<iframe src="demo_iframe.htm" style="height:200px;width:300px;" title="
Iframe Example"></iframe>
```

Iframe - Remove the Border

By default, an iframe has a border around it.

To remove the border, add the `style` attribute and use the CSS `border` property:

Example

```
<iframe src="demo_iframe.htm" style="border:none;" title="Iframe
Example"></iframe>
```

With CSS, you can also change the size, style and color of the iframe's border:

Example

```
<iframe src="demo_iframe.htm" style="border:2px solid
red;" title="Iframe Example"></iframe>
```

Iframe - Target for a Link

An iframe can be used as the target frame for a link.

The `target` attribute of the link must refer to the `name` attribute of the iframe:

Example

```
<iframe src="demo_iframe.htm" name="iframe_a" title="Iframe
Example"></iframe>

<p><a href="https://www.w3schools.com" target="iframe_a">W3Schools.com<
/a></p>
```

Chapter Summary

- The HTML `<iframe>` tag specifies an inline frame
- The `src` attribute defines the URL of the page to embed
- Always include a `title` attribute (for screen readers)
- The `height` and `width` attributes specify the size of the iframe
- Use `border:none;` to remove the border around the iframe

Exercise:

Create an iframe with a URL address that goes to https://www.`facebook`.com.

```
<iframe ="https://www.facebook.com"></iframe>
```

HTML iframe Tag

Tag	Description
<iframe>	Defines an inline frame

Exercise:

Create an iframe with a URL address that goes to https://www.w3schools.com.

```
<iframe ="https://www.w3schools.com"></iframe>
```

HTML JavaScript

JavaScript makes HTML pages more dynamic and interactive.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<button type="button"
```

```
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>

<p id="demo"></p>

</body>
</html>
```

The HTML <script> Tag

The HTML `<script>` tag is used to define a client-side script (JavaScript).

The `<script>` element either contains script statements, or it points to an external script file through the `src` attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

To select an HTML element, JavaScript most often uses the `document.getElementById()` method.

This JavaScript example writes "Hello JavaScript!" into an HTML element with id="demo":

Example

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
```

A Taste of JavaScript

Here are some examples of what JavaScript can do:

Example

JavaScript can change content:

```
document.getElementById("demo").innerHTML = "Hello JavaScript!";
```

Example

JavaScript can change styles:

```
document.getElementById("demo").style.fontSize = "25px";
document.getElementById("demo").style.color = "red";
document.getElementById("demo").style.backgroundColor = "yellow";
```

Example

JavaScript can change attributes:

```
document.getElementById("image").src = "picture.gif";
```

The HTML <noscript> Tag

The HTML `<noscript>` tag defines an alternate content to be displayed to users that have disabled scripts in their browser or have a browser that doesn't support scripts:

Example

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
<noscript>Sorry, your browser does not support JavaScript!</noscript>
```

Exercise:

Use JavaScript to change the HTML content of the `<p>` element to "Hello World!".

```
<body>

<p id="demo">Hi.</p>

<script>
document.("demo").innerHTML = "Hello World!";
</script>

</body>
```

HTML Script Tags

Tag	Description
<script>	Defines a client-side script
<noscript>	Defines an alternate content for users that do not support client-side scripts

HTML File Paths

File Path Examples

Path	Description
<img src="picture.jpg">	The "picture.jpg" file is located in the same folder as the current page
<img src="images/picture.jpg">	The "picture.jpg" file is located in the images folder in the current folder
<img src="/images/picture.jpg">	The "picture.jpg" file is located in the images folder at the root of the current web
<img src="../picture.jpg">	The "picture.jpg" file is located in the folder one level up from the current folder

HTML File Paths

A file path describes the location of a file in a web site's folder structure.

File paths are used when linking to external files, like:

- Web pages
- Images
- Style sheets
- JavaScripts

Absolute File Paths

An absolute file path is the full URL to a file:

Example

```
<img src="https://www.w3schools.com/images/picture.jpg" alt="Mountain">
```

Relative File Paths

A relative file path points to a file relative to the current page.

In the following example, the file path points to a file in the images folder located at the root of the current web:

Example

```
<img src="/images/picture.jpg" alt="Mountain">
```

In the following example, the file path points to a file in the images folder located in the current folder:

Example

```
<img src="images/picture.jpg" alt="Mountain">
```

In the following example, the file path points to a file in the images folder located in the folder one level up from the current folder:

Example

```
<img src="../images/picture.jpg" alt="Mountain">
```

Best Practice

It is best practice to use relative file paths (if possible).

When using relative file paths, your web pages will not be bound to your current base URL. All links will work on your own computer (localhost) as well as on your current public domain and your future public domains.

Responsive Web Design - Introduction

What is Responsive Web Design?

Responsive web design makes your web page look good on all devices.

Responsive web design uses only HTML and CSS.

Responsive web design is not a program or a JavaScript.

Designing For The Best Experience For All Users

Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device.

Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:

It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

Don't worry if you don't understand the example below, we will break down the code, step-by-step, in the next chapters:

```
<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-
scale=1.0">

<style>

* {

  box-sizing: border-box;

}


.row::after {

  content: "";

  clear: both;

  display: table;

}


[class*="col-"] {

  float: left;

  padding: 15px;

}


html {

  font-family: "Lucida Sans", sans-serif;
}

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}

.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.menu li:hover {
  background-color: #0099cc;
}
```

```
.aside {
  background-color: #33b5e5;
  padding: 15px;
  color: #ffffff;
  text-align: center;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.footer {
  background-color: #0099cc;
  color: #ffffff;
  text-align: center;
  font-size: 12px;
  padding: 15px;
}

/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}

@media only screen and (min-width: 600px) {
  /* For tablets: */
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-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>

</head>

<body>


<div class="header">

  <h1>Chania</h1>

</div>


<div class="row">

  <div class="col-3 col-s-3 menu">

    <ul>

      <li>The Flight</li>

      <li>The City</li>

      <li>The Island</li>

      <li>The Food</li>

    </ul>

  </div>


  <div class="col-6 col-s-9">

    <h1>The City</h1>

    <p>Chania is the capital of the Chania region on the island of Crete.
The city can be divided in two parts, the old town and the modern
city.</p>

  </div>


  <div class="col-3 col-s-12">

    <div class="aside">
```

```
      <h2>What?</h2>

      <p>Chania is a city on the island of Crete.</p>

      <h2>Where?</h2>

      <p>Crete is a Greek island in the Mediterranean Sea.</p>

      <h2>How?</h2>

      <p>You can reach Chania airport from all over Europe.</p>

    </div>

  </div>

</div>


<div class="footer">

  <p>Resize the browser window to see how the content respond to the resizing.</p>

</div>


</body>

</html>
```

Responsive Web Design - The Viewport

What is The Viewport?

The viewport is the user's visible area of a web page.

The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

This was not perfect!! But a quick fix.

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the `<meta>` tag.

You should include the following `<meta>` viewport element in all your web pages:

```html
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

This gives the browser instructions on how to control the page's dimensions and scaling.

The `width=device-width` part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The `initial-scale=1.0` part sets the initial zoom level when the page is first loaded by the browser.

Size Content to The Viewport

Users are used to scroll websites vertically on both desktop and mobile devices - but not horizontally!

So, if the user is forced to scroll horizontally, or zoom out, to see the whole web page it results in a poor user experience.

Some additional rules to follow:

1. Do NOT use large fixed width elements - For example, if an image is displayed at a width wider than the viewport it can cause the viewport to scroll horizontally. Remember to adjust this content to fit within the width of the viewport.

2. Do NOT let the content rely on a particular viewport width to render well - Since screen dimensions and width in CSS pixels vary widely between devices, content should not rely on a particular viewport width to render well.

3. Use CSS media queries to apply different styling for small and large screens - Setting large absolute CSS widths for page elements will cause the element to be too wide for the viewport on a smaller device. Instead, consider using relative width values, such as width: 100%. Also, be careful of using large absolute positioning values. It may cause the element to fall outside the viewport on small devices.

Responsive Web Design - Media Queries

What is a Media Query?

Media query is a CSS technique introduced in CSS3.

It uses the `@media` rule to include a block of CSS properties only if a certain condition is true.

Example

If the browser window is 600px or smaller, the background color will be lightblue:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {
  background-color: lightgreen;
}
```

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
</style>
</head>
<body>

<p>Resize the browser window. When the width of this document is 600 pixels or
less, the background-color is "lightblue", otherwise it is "lightgreen".</p>

</body>
</html>
```

Add a Breakpoint

Earlier in this tutorial we made a web page with rows and columns, and it was responsive, but it did not look good on a small screen.

Media queries can help with that. We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        /* Common CSS for all screen sizes */
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
        }

        header {
            background-color: #333;
            color: #fff;
            text-align: center;
            padding: 20px;
        }

        section {
            padding: 20px;
            display: flex;
            flex-wrap: wrap;
        }

        aside {
            background-color: #f0f0f0;
            padding: 10px;
            float: left;
            width: 30%;
        }

        aside ul {
            list-style-type: none;
            padding: 0;
            display: flex;
            justify-content: space-between;
        }

        aside li {
            margin-bottom: 10px;
        }

        main {
            float: left;
            width: 70%;
        }

        /* Media Query for screens smaller than 600px */
        @media screen and (max-width: 600px) {
            header {
                font-size: 20px;
```

```
            }
            aside, main {
                width: 100%;
            }
              aside ul {
                flex-direction: column;
            }
        }

        /* Media Query for screens between 601px and 1024px */
        @media screen and (min-width: 601px) and (max-width: 1024px) {
            header {
                font-size: 24px;
            }
            aside, main {
                width: 100%;
            }
        }

        /* Media Query for screens larger than 1024px */
        @media screen and (min-width: 1025px) {
            header {
                font-size: 28px;
            }
            aside {
                width: 30%;
            }
            main {
                width: 70%;
            }
        }
    </style>
</head>
<body>
    <header>
        Responsive Design Example
    </header>
    <section>
        <aside>
            <h2>Navigation</h2>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Services</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </aside>
        <main>
            <h1>Welcome to our website</h1>
            <p>This is a simple example of a responsive design using HTML and CSS
with media queries. The header text size changes based on the screen width.</p>
            <h2>About Us</h2>
            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac metus
vitae orci convallis volutpat. Aenean at lorem sit amet nulla euismod congue. Nulla
facilisi.</p>
            <h2>Services</h2>
            <p>We offer a wide range of services to meet your needs. Our services
include web design, development, and digital marketing.</p>
        </main>
    </section>
</body>
</html>
```

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

```
<p id="demo"></p>

<script>
   // Function to compute the product of p1 and p2
     function myFunction(p1, p2) {
           return p1 * p2;
     }
     let result = myFunction(4, 3);
     document.getElementById("demo").innerHTML = result;
</script>
```

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(parameter1, parameter2, ...)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this lesson.

Function Return

When JavaScript reaches a `return` statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);

function myFunction(a, b) {
// Function returns the product of a and b
  return a * b;
}
```

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

You can use the same code with different arguments, to produce different results.

The () Operator

The () operator invokes (calls) the function:

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius(77);

document.getElementById("demo").innerHTML = value;
```

Accessing a function with incorrect parameters can return an incorrect answer:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius();
```

Accessing a function without () returns the function and not the function result:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
```

```
let value = toCelsius;
```

Note

As you see from the examples above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Exercise:

Execute the function named `myFunction`.

```
function myFunction() {
  alert("Hello World!");
};
```

JavaScript Control Structures

JavaScript if, else, and else if

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

The `switch` statement is described in the next chapter.

The if Statement

Use the `if` statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example

Make a "Good day" greeting if the hour is less than 18:00:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript if</h2>

<p>Display "Good day!" if the hour is less than 18:00:</p>

<p id="demo">Good Evening!</p>
```

```
<script>
if (new Date().getHours() < 10) {
  document.getElementById("demo").innerHTML = "Good day!";
}
</script>

</body>
</html>
```

The else Statement

Use the `else` statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Example

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

<p id="demo"></p>

<script>
const hour = new Date().getHours();
let greeting;

if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}

document.getElementById("demo").innerHTML = greeting;
</script>

</body>
</html>
```

The else if Statement

Use the `else if` statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and
condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and
condition2 is false
}
```

Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Exercise:

Fix the `if` statement to alert "Hello World" if `x` is greater than `y`.

```
if x > y
  alert("Hello World");
```

JavaScript Switch Statement

The `switch` statement is used to perform different actions based on different conditions.

The JavaScript Switch Statement

Use the `switch` statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Example

The getDay() method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
let day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>

</body>
</html>
```

The result of day will be:

Thursday

The break Keyword

When JavaScript reaches a `break` keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

Note: If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

The default Keyword

The `default` keyword specifies the code to run if there is no case match:

Example

The `getDay()` method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

The result of text will be:

Looking forward to the Weekend

The default case does not have to be the last case in a switch block:

Example

```
switch (new Date().getDay()) {
  default:
    text = "Looking forward to the Weekend";
    break;
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
}
```

If default is not the last case in the switch block, remember to end the default case with a break.

Common Code Blocks

Sometimes you will want different switch cases to use the same code.

In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

Example

```
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Switching Details

If multiple cases matches a case value, the **first** case is selected.

If no matching cases are found, the program continues to the **default** label.

If no default label is found, the program continues to the statement(s) **after the switch**.

Strict Comparison

Switch cases use **strict** comparison (===).

The values must be of the same type to match.

A strict comparison can only be true if the operands are of the same type.

In this example there will be no match for x:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
let x = "0";

switch (x) {
  case 0:
    text = "Off";
    break;
  case 1:
    text = "On";
    break;
  default:
    text = "No value found";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
```

```
</html>
```

Exercise:

Create a switch statement that will alert "Hello" if fruits is "banana", and "Welcome" if fruits is "apple".

```
______(fruits) {
 ____ "Banana":
    alert("Hello")
    break;
  ____"Apple":
    alert("Welcome")
    break;
}
```

JavaScript For Loop

Loops can execute a block of code a number of times.

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>JavaScript For Loop</h2>

<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

let text = "";
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Different Kinds of Loops

JavaScript supports different kinds of loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `for/of` - loops through the values of an iterable object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

The For Loop

The `for` statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Loop</h2>

<p id="demo"></p>

<script>
let text = "";

for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

Expression 1

Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case. JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

Example

```
<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let i, len, text;
for (i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
```

And you can omit expression 1 (like when your values are set before the loop starts):

Example

```
let i = 2;
let len = cars.length;
let text = "";
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again. If it returns false, the loop will end.

If you omit expression 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Expression 3

Often expression 3 increments the value of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 3 is optional.

Expression 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

Example

```
let i = 0;
let len = cars.length;
let text = "";
for (; i < len; ) {
  text += cars[i] + "<br>";
  i++;
}
```

Loop Scope

Using `var` in a loop:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript var</h2>

<p id="demo"></p>

<script>
var i = 5;
for (var i = 0; i < 10; i++) {
  // some statements
}
document.getElementById("demo").innerHTML = i;
</script>

</body>
</html>
```

Using `let` in a loop:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript let</h2>

<p id="demo"></p>

<script>
let i = 5;
for (let i = 0; i < 10; i++) {
  // some statements
}
document.getElementById("demo").innerHTML = i;
</script>

</body>
</html>
```

In the first example, using `var`, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using `let`, the variable declared in the loop does not redeclare the variable outside the loop.

When `let` is used to declare the i variable in a loop, the i variable will only be visible within the loop.

For/Of and For/In Loops

The `for/in` loop and the `for/of` loop are explained in the next chapter.

While Loops

The `while` loop and the `do/while` are explained in the next chapters.

Exercise:

Create a loop that runs from 0 to 9.

JavaScript For In

The For In Loop

The JavaScript `for in` statement loops through the properties of an Object:

Syntax

```
for (key in object) {
  // code block to be executed
}
```

Example

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
document.getElementById("demo").innerHTML = txt;
```

Example Explained

- The **for in** loop iterates over a **person** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key
- The value of the key is **person[x]**

For In Over Arrays

The JavaScript `for in` statement can also loop over the properties of an Array:

Syntax

```
for (variable in array) {
  code
}
```

Example

```
  const numbers = [45, 4, 9, 16, 25];

  let txt = "";
  for (let x in numbers) {
    txt += numbers[x];
  }
document.getElementById("demo").innerHTML = txt;
```

Do not use **for in** over an Array if the index **order** is important.

The index order is implementation-dependent, and array values may not be accessed in the order you expect.

It is better to use a **for** loop, a **for of** loop, or **Array.forEach()** when the order is important.

Array.forEach()

The `forEach()` method calls a function (a callback function) once for each array element.

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
```

```
<h2>The forEach() Method</h2>

<p>Call a function once for each array element:</p>

<p id="demo"></p>

<script>
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value, index, array) {
  txt += value + "<br>";
}
</script>

</body>
</html>
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter. It can be rewritten to:

Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);

function myFunction(value) {
  txt += value;
}
```

JavaScript For Of

The For Of Loop

The JavaScript `for of` statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax

```
for (variable of iterable) {
  // code block to be executed
}
```

variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with `const`, `let`, or `var`.

iterable - An object that has iterable properties.

Looping over an Array

```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}
```

Looping over a String

Example

```
let language = "JavaScript";

let text = "";
for (let x of language) {
text += x;
}
```

JavaScript While Loop

Loops can execute a block of code as long as a specified condition is true.

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
  // code block to be executed
}
```

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript While Loop</h2>

<p id="demo"></p>

<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>
```

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

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The Do While Loop

The `do while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
  // code block to be executed
}
while (condition);
```

Example

The example below uses a `do while` loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a `for` loop to collect the car names from the cars array:

Example

```
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let i = 0;
let text = "";

for (;cars[i];) {
  text += cars[i];
  i++;
}
```

The loop in this example uses a `while` loop to collect the car names from the cars array:

Example

```
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let i = 0;
let text = "";

while (cars[i]) {
  text += cars[i];
  i++;
}
```

Exercise:

Create a loop that runs as long as `i` is less than 10.

Week 8:

HTML Entities

Reserved characters in HTML must be replaced with entities:

- < (less than) = **&lt;**
- > (greather than) = **&gt;**

HTML Character Entities

Some characters are reserved in HTML.

If you use the less than (<) or greater than (>) signs in your HTML text, the browser might mix them with tags.

Entity names or entity numbers can be used to display reserved HTML characters.

Entity names look like this:

> &*entity_name*;

Entity numbers look like this:

> &#*entity_number*;

To display a less than sign (<) we must write: **&lt;** or **&#60;**

Entity names are easier to remember than entity numbers.

Non-breaking Space

A commonly used HTML entity is the non-breaking space: **&nbsp;**

A non-breaking space is a space that will not break into a new line.

Two words separated by a non-breaking space will stick together (not break into a new line). This is handy when breaking the words might be disruptive.

Examples:

- § 10
- 10 km/h
- 10 PM

Another common use of the non-breaking space is to prevent browsers from truncating spaces in HTML pages.

If you write 10 spaces in your text, the browser will remove 9 of them. To add real spaces to your text, you can use the **&nbsp;** character entity.

The non-breaking hyphen (&#8209;) is used to define a hyphen character (‑) that does not break into a new line.

Some Useful HTML Character Entities

Result	Description	Name	Number
	non-breaking space	&nbsp;	&#160;
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark	&apos;	&#39;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;

¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	trademark	&reg;	&#174;

Note

Entity names are case sensitive.

HTML Symbols

Symbols or letters that are not present on your keyboard can be added to HTML using entities.

HTML Symbol Entities

HTML entities were described in the previous chapter.

Many mathematical, technical, and currency symbols, are not present on a normal keyboard.

To add such symbols to an HTML page, you can use the entity name or the entity number (a decimal or a hexadecimal reference) for the symbol:

Example

Display the euro sign:

```
<p>I will display &euro;</p>
<p>I will display &#8364;</p>
<p>I will display &#x20AC;</p>
```

Will display as:

I will display €
I will display €
I will display €

Some Mathematical Symbols Supported by HTML

Char	Number	Entity	Description
∀	&#8704;	&forall;	For all
∂	&#8706;	&part;	Partial differential
∃	&#8707;	&exist;	There exists
∅	&#8709;	&empty;	Empty sets
∇	&#8711;	&nabla;	Nabla
∈	&#8712;	&isin;	Element of
∉	&#8713;	&notin;	Not an element of
∋	&#8715;	&ni;	Contains as member
∏	&#8719;	&prod;	N-ary product
∑	&#8721;	&sum;	N-ary summation

Make research on other HTML entities and Symbols

HTML Uniform Resource Locators

A URL is another word for a web address.

A URL can be composed of words (e.g. w3schools.com), or an Internet Protocol (IP) address (e.g. 192.68.20.50).

Most people enter the name when surfing, because names are easier to remember than numbers.

URL - Uniform Resource Locator

Web browsers request pages from web servers by using a URL.

A Uniform Resource Locator (URL) is used to address a document (or other data) on the web.

A web address like https://www.facebook.com follows these syntax rules:

```
scheme://prefix.domain:port/path/filename
```

Explanation:

- **scheme** - defines the **type** of Internet service (most common is **http or https**)
- **prefix** - defines a domain **prefix** (default for http is **www**)
- **domain** - defines the Internet **domain name** (like w3schools.com)
- **port** - defines the **port number** at the host (default for http is **80**)
- **path** - defines a **path** at the server (If omitted: the root directory of the site)
- **filename** - defines the name of a document or resource

Common URL Schemes

The table below lists some common schemes:

Scheme	Short for	Used for
http	HyperText Transfer Protocol	Common web pages. Not encrypted
https	Secure HyperText Transfer Protocol	Secure web pages. Encrypted
ftp	File Transfer Protocol	Downloading or uploading files
file		A file on your computer

URL Encoding

URLs can only be sent over the Internet using the ASCII character-set. If a URL contains characters outside the ASCII set, the URL has to be converted.

URL encoding converts non-ASCII characters into a format that can be transmitted over the Internet.

URL encoding replaces non-ASCII characters with a "%" followed by hexadecimal digits.

URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign, or %20.

ASCII Encoding Examples

Your browser will encode input, according to the character-set used in your page.

The default character-set in HTML5 is UTF-8.

Character	**From Windows-1252**	**From UTF-8**
€	%80	%E2%82%AC
£	%A3	%C2%A3
©	%A9	%C2%A9
®	%AE	%C2%AE
À	%C0	%C3%80
Á	%C1	%C3%81
Â	%C2	%C3%82
Ã	%C3	%C3%83
Ä	%C4	%C3%84
Å	%C5	%C3%85

JavaScript HTML DOM

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

What You Will Learn

In the next chapters of this tutorial you will learn:

- How to change the content of HTML elements
- How to change the style (CSS) of HTML elements
- How to react to HTML DOM events
- How to add and delete HTML elements

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

JavaScript - HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`:

Example

```
<html>
<body>

<p id="demo"></p>

<script>
```

```
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

In the example above, `getElementById` is a **method**, while `innerHTML` is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the `id` of the element.

In the example above the `getElementById` method used `id="demo"` to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the `innerHTML` property.

The `innerHTML` property is useful for getting or replacing the content of HTML elements.

The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an `onclick` attribute (with code), is added to a `<button>` element:

Example

```
<button onclick="document.getElementById('demo').innerHTML =
Date()">The time is?</button>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using `this.innerHTML`):

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Example

```
<button onclick="displayDate()">The time is?</button>
```

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

Exercise:

The `<button>` element should do something when someone clicks on it. Try to fix it!

```
<button    ="alert('Hello')">Click me.</button>
```

JavaScript HTML DOM Document

The HTML DOM document object is the owner of all other objects in your web page.

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element

Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(id).onclick = function(){code}	Adding event handler code to an onclick event

Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Deprecated	1

document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete.	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3

document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

JavaScript HTML DOM Elements

This page teaches you how to find and access HTML elements in an HTML page.

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are several ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with `id="intro"`:

Example

```
const element = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in element).

If the element is not found, element will contain `null`.

Finding HTML Elements by Tag Name

This example finds all `<p>` elements:

Example

```
const element = document.getElementsByTagName("p");
```

This example finds the element with `id="main"`, and then finds all `<p>` elements inside `"main"`:

Example

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use `getElementsByClassName()`.

This example returns a list of all elements with `class="intro"`.

Example

```
const x = document.getElementsByClassName("intro");
```

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.

This example returns a list of all `<p>` elements with `class="intro"`.

Example

```
const x = document.querySelectorAll("p.intro");
```

Finding HTML Elements by HTML Object Collections

This example finds the form element with `id="frm1"`, in the forms collection, and displays all element values:

Example

```
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length; i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

The following HTML objects (and object collections) are also accessible:

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title

Exercise:

Use the `getElementById` method to find the `<p>` element, and change its text to "Hello".

JavaScript HTML DOM - Changing HTML

The HTML DOM allows JavaScript to change the content of HTML elements.

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the `innerHTML` property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML = new HTML
```

This example changes the content of a `<p>` element:

Example

```html
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains a `<p>` element with `id="p1"`
- We use the HTML DOM to get the element with `id="p1"`
- A JavaScript changes the content (`innerHTML`) of that element to "New text!"

This example changes the content of an `<h1>` element:

Example

```html
<!DOCTYPE html>
<html>
<body>

<h1 id="id01">Old Heading</h1>

<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>
```

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

Example explained:

- The HTML document above contains an `<h1>` element with `id="id01"`
- We use the HTML DOM to get the element with `id="id01"`
- A JavaScript changes the content (`innerHTML`) of that element to "New Heading"

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

```
document.getElementById(id).attribute = new value
```

This example changes the value of the src attribute of an `<img>` element:

Example

```
<!DOCTYPE html>
<html>
<body>

<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an `<img>` element with `id="myImage"`
- We use the HTML DOM to get the element with `id="myImage"`

- A JavaScript changes the `src` attribute of that element from "smiley.gif" to "landscape.jpg"

Dynamic HTML content

JavaScript can create dynamic HTML content:

Date : Thu Nov 02 2023 20:45:16 GMT+0000 (Greenwich Mean Time)

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
document.getElementById("demo").innerHTML = "Date : " +
Date(); </script>

</body>
</html>
```

document.write()

In JavaScript, `document.write()` can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Bla bla bla</p>

<script>
document.write(Date());
</script>

<p>Bla bla bla</p>
```

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

Never use `document.write()` after the document is loaded. It will overwrite the document.

Exercise:

Use HTML DOM to change the value of the image's src attribute.

```
<img id="image" src="smiley.gif">

<script>
document.getElementById("image") = "pic_mountain.jpg";
</script>
```

JavaScript Forms

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php" onsubmit="return
validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

JavaScript Can Validate Numeric Input

JavaScript is often used to validate numeric input:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Validation</h2>

<p>Please input a number between 1 and 10:</p>

<input id="numb">

<button type="button" onclick="myFunction()">Submit</button>

<p id="demo"></p>

<script>
function myFunction() {
  // Get the value of the input field with id="numb"
  let x = document.getElementById("numb").value;
  // If x is Not a Number or less than one or greater than 10
  let text;
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
```

```
</html>
```

Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:

If a form field (fname) is empty, the `required` attribute prevents this form from being submitted:

HTML Form Example

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```

Automatic HTML form validation does not work in Internet Explorer 9 or earlier.

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

- Constraint validation **HTML Input Attributes**
- Constraint validation **CSS Pseudo Selectors**
- Constraint validation **DOM Properties and Methods**

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

Constraint Validation CSS Pseudo Selectors

Selector	Description
:disabled	Selects input elements with the "disabled" attribute specified
:invalid	Selects input elements with invalid values
:optional	Selects input elements with no "required" attribute specified
:required	Selects input elements with the "required" attribute specified
:valid	Selects input elements with valid values

JavaScript HTML DOM - Changing CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = new style
```

The following example changes the style of a `<p>` element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

</body>
</html>
```

Using Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

You will learn more about events in the next chapter of this tutorial.

This example changes the style of the HTML element with `id="id1"`, when the user clicks a button:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

Exercise:

Change the text color of the `<p>` element to "red".

```
<p id="demo"></p>

<script>
document.getElementById("demo")         = "red";
</script>
```

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events:

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

```
onclick=JavaScript
```

Examples of HTML events:

- When a user clicks the mouse

- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the `<h1>` element is changed when a user clicks on it:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

</body>
</html>
```

In this example, a function is called from the event handler:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

<p id="demo"></p>

</body>
</html>
```

In the example above, a function named `displayDate` will be executed when the button is clicked.

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick = displayDate;
```

Example`</script>`

In the example above, a function named `displayDate` is assigned to an HTML element with the `id="myBtn"`.

The function will be executed when the button is clicked.

The onload and onunload Events

The `onload` and `onunload` events are triggered when the user enters or leaves the page.

The `onload` event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The `onload` and `onunload` events can be used to deal with cookies.

```
<body onload="checkCookies()">
```

The onchange Event

The `onchange` event is often used in combination with validation of input fields.

Below is an example of how to use the onchange.
The `upperCase()` function will be called when a user changes the content of an input field.

Example

```
<input type="text" id="fname" onchange="upperCase()">
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML Events</h2>
Enter your name: <input type="text" id="fname"
onchange="upperCase()">
<p>When you leave the input field, a function is triggered which
transforms the input text to upper case.</p>
```

```
<script>
function upperCase() {
  const x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>

</body>
</html>
```

The onmouseover and onmouseout Events

The `onmouseover` and `onmouseout` events can be used to trigger a function when the user mouses over, or out of, an HTML element:

```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-
color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
  obj.innerHTML = "Thank You"
}

function mOut(obj) {
  obj.innerHTML = "Mouse Over Me"
}
</script>

</body>
</html>
```

The onmousedown, onmouseup and onclick Events

The `onmousedown`, `onmouseup`, and `onclick` events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

```
<!DOCTYPE html>
```

```
<html>
<body>

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-
color:#D94A38;width:90px;height:20px;padding:40px;">
Click Me</div>

<script>
function mDown(obj) {
  obj.style.backgroundColor = "#1ec5e5";
  obj.innerHTML = "Release Me";
}

function mUp(obj) {
  obj.style.backgroundColor="#D94A38";
  obj.innerHTML="Thank You";
}
</script>

</body>
</html>
```

JavaScript HTML DOM EventListener

The addEventListener() method

Example

Add an event listener that fires when a user clicks a button:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a
click event to a button.</p>

<button id="myBtn">Try it</button>
```

```
<p id="demo"></p>

<script>
document.getElementById("myBtn").addEventListener("click",
displayDate);

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

The `addEventListener()` method attaches an event handler to the specified element.

The `addEventListener()` method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The `addEventListener()` method makes it easier to control how the event reacts to bubbling.

When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the `removeEventListener()` method.

Syntax

```
element.addEventListener(event, function, useCapture);
```

The first parameter is the type of the event (like "`click`" or "`mousedown`" or any other HTML DOM Event.)

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Note that you don't use the "on" prefix for the event; use "`click`" instead of "`onclick`".

Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

You can also refer to an external "named" function:

Example

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
```

Add Many Event Handlers to the Same Element

The `addEventListener()` method allows you to add many events to the same element, without overwriting existing events:

Example

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

You can add events of different types to the same element:

Example

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

Add an Event Handler to the window Object

The `addEventListener()` method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that support events, like the `xmlHttpRequest` object.

Example

Add an event listener that fires when a user resizes the window:

```
window.addEventListener("resize", function(){
  document.getElementById("demo").innerHTML = sometext;
});
```

Passing Parameters

When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

Example

```
element.addEventListener("click", function(){ myFunction(p1, p2); });
```

Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?

In bubbling the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

In capturing the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

```
addEventListener(event, function, useCapture);
```

The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

Example

```
document.getElementById("myP").addEventListener("click",
myFunction, true);
document.getElementById("myDiv").addEventListener("click",
myFunction, true);
```

The removeEventListener() method

The `removeEventListener()` method removes event handlers that have been attached with the addEventListener() method:

Example

```
element.removeEventListener("mousemove", myFunction);
```

Exercise:

Use the `eventListener` to assign an onclick event to the `<button>` element.

```
<button id="demo"></button>

<script>
document.getElementById("demo").        ("     ", myFunction);
</script>
```