**Cascading Style Sheets (CSS)**

<https://www.w3schools.com/css/default.asp>

<https://developer.mozilla.org/en-US/docs/Web/CSS>

**CSS, What?**

CSS is a language that describes the style of an HTML document, i.e., the design, layout and variations in display for different devices and screen sizes. It describes **how** HTML elements should be displayed on the Web Browser.

**CSS, Why?**

HTML was created to describe the content of a web page, but it was NEVER intended to contain tags for formatting a web page! For example, consider the following simple Web page consisting of a heading and a paragraph.

|  |
| --- |
| <body>  <h1>This is a heading</h1>  <p>This is a paragraph.</p>  </body> |

Looking at this page, we know what is to be displayed on the screen. But there is NO information as to HOW the elements will be displayed:

* What is background color? Are we going to use a background image?
* What’s the font color?
* What’s the font-family?
* What are the margins and padding from the left, right, top or bottom?
* Will any of the element have a border? If yes, what’s the border style (solid-dashed), width (how many pixels) etc.
* Will both elements (heading and paragraph) have the same formatting style or will they have different formatting?

There are a ton of formatting questions that must be answered for each HTML element. The question is, how do we specify these styles? And even better: Can we design a general style file and use it to format all of our web pages in our site? That’s what CSS is all about. The idea with CSS is to separate content from presentation. HTML describes the content (WHAT) and CSS specifies the presentation (HOW). This separation of concerns allows us to have very clean, consistent and responsive Web pages. Responsive means that the Web page presentation adjusts itself automatically to different screen sizes (Desktop, tables and smart phones).

**CSS, How?**

There are three ways of specifying styles for an HTML page:

* Inline CSS
* Internal CSS
* External CSS

**Inline CSS**

We have already seen how inline CSS is specified using the “style” attribute:

|  |
| --- |
| <body>  <h1 style="background-color: black; color: white">This is a heading</h1>  <p style="color: red; font-family: Arial, Helvetica, sans-serif">This is a paragraph.</p>  </body> |

An inline style mixes the content with presentation and is NOT recommended. Use this method sparingly.

Look at: 02-CSS/01-InlineCSS

Instead of using inline CSS, the recommended way is to separate the presentation from content, and write CSS somewhere else. In one case, we write CSS in a separate section (inside head) within the same HTML file called the internal CSS, and in the other case, we write CSS in a separate file and include it from all HTML file where we would like to use the style called the external CSS. But before we talk about these two methods, we must clear up one issue: When we separate HTML elements and their corresponding CSS, how do we select or refer to the HTML element from within CSS? That is, how does the browser know which styles in CSS to apply to which elements in HTML? Considering our example, how do we specify in CSS that style=”background-color: black; color: white” must be applied to <h1>, and style="color: red; font-family: Arial, Helvetica, sans-serif"> must be applied to <p>? This is where CSS selectors come into play.

**CSS Rules and Selectors**

A CSS rule-set consists of a selector and a declaration block:



The selector points to the HTML element you want to style. The declaration block contains one or more declarations separated by semicolons. Each declaration includes a CSS property name and a value, separated by a colon. A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces. In our example, we will have the following CSS rules:

|  |
| --- |
| h1 {  background-color: black;  color: white;  }  p {  color: red;  font-family: Arial, Helvetica, sans-serif;  } |

Given these CSS rules that have been separated from the HTML, there are two methods to associate them with the HTML: Internal CSS and External CSS

**Internal CSS**

The internal style is defined inside the <style> element, inside the head section. It may be used if one single HTML page has a unique style.

|  |
| --- |
| <head>  <style>  h1 {  background-color: black;  color: white;  }  p {  color: red;  font-family: Arial, Helvetica, sans-serif;  }  </style>  </head>  <body>  <h1>This is a heading</h1>  <p>This is a paragraph.</p>  </body> |

As you can see, we simply copied the internal style attributes from the HTML elements into CSS rules and placed them inside the <style> tag in the head section.

Look at: 02-CSS/02-InternalCSS

**External CSS**

Finally, the usual practice is to copy the CSS rules into a separate style file with a .css extension and include this style file in all HTML files that uses it for styling. Thus, each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section. This method allows writing up a single style file to format many pages in the same Web site, which not only allows easy formatting but also enables consistency among different pages on the site, which is also very important for good user experience.

Here is mystyle.css that contains the CSS rules for our example:

|  |
| --- |
| h1 {  background-color: black;  color: white;  }  p {  color: red;  font-family: Arial, Helvetica, sans-serif;  } |

Here is how we include this CSS file from within our HTML file:

|  |
| --- |
| <head>  <title>External CSS Example</title>  <link rel="stylesheet" type="text/css" href="mystyle.css">  </head>  <body>  <h1>This is a heading</h1>  <p>This is a paragraph.</p>  </body> |

Look at: 02-CSS/03-ExternalCSS

**Can I Use a Certain CSS Property?**

Before we go any further, it is important to mention that not all browsers support all CSS properties. To see which browsers support a CSS property, you can go to <https://caniuse.com/> and type in the name of the property you want to use and see which browsers support it. It is always good practice to make use of CSS properties supported by a wide variety of browsers.

**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-element selectors (select and style a part of an element)
* Attribute selectors (select elements based on an attribute or attribute value)

**CSS Simple Selectors**

1. **The CSS Universal Selector**

The universal selector (\*) selects all HTML elements on the page. The CSS rule below will affect every HTML element on the page:

|  |
| --- |
| \* {  text-align: center;  color: blue;  } |

1. **The CSS HTML element Selector**

The element selector selects HTML elements based on the element name. Here is an example, which selects all <p> elements on the page and make them center-aligned, with a red text color:

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

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

|  |
| --- |
| #para1 {  text-align: left;  color: blue;  } |

1. **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. In the following example, all HTML elements with class="center" will be green and center-aligned:

|  |
| --- |
| .center {  text-align: center;  color: green;  } |

You can also specify that only specific HTML elements should be affected by a class. In the following example, only <p> elements with class="center" will be center-aligned:

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

**Grouping CSS Selectors**

It is possible to group all CSS rules 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.

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

**Summary of CSS Simple Selectors**

Here is a summary of all CSS Simple Selectors:

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Description** |
| \* | \* | Selects all elements |
| #id | #firstname | Selects the element with id="firstname" |
| .class | .intro | Selects all elements with class="intro" |
| element | p | Selects all <p> elements |
| element, element,.. | div, p | Selects all <div> elements and all <p> elements |

Look at: 02-CSS/04-SimpleSelectors

**Combinator Selectors**

A combinator is something that explains the relationship between the selectors. A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator. There are four different combinators in CSS:

* descendant selector (space)
* child selector (>)
* adjacent sibling selector (+)
* general sibling selector (~)

1. **Descendant Selector**

The descendant selector matches all elements that are descendants of a specified element. The following example selects all <p> elements inside <div> elements:

|  |
| --- |
| div p {  background-color: yellow;  } |

1. **Child Selector**

The child selector selects all elements that are the children of a specified element. The following example selects all <p> elements that are children of a <div> element:

|  |
| --- |
| div > p {  background-color: yellow;  } |

1. **Adjacent Sibling Selector**

The adjacent sibling selector selects all elements that are the adjacent siblings of a specified element. Sibling elements must have the same parent element, and "adjacent" means "immediately following". The following example selects all <p> elements that are placed immediately after <div> elements:

|  |
| --- |
| div + p {  background-color: yellow;  } |

1. **General Sibling Selector**

The general sibling selector selects all elements that are siblings of a specified element. The following example selects all <p> elements that are siblings of <div> elements:

|  |
| --- |
| div ~ p {  background-color: yellow;  } |

**Summary of CSS Combinator Selectors**

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Example description** |
| element element | div p | Selects all <p> elements inside <div> elements |
| element>element | div > p | Selects all <p> elements where the parent is a <div> element |
| element+element | div + p | Selects all <p> elements that are placed immediately after <div> elements |
| element1~element2 | p ~ ul | Selects every <ul> element that are preceded by a <p> element |

**CSS 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 moves the mouse over it (hover)
* Style visited and unvisited links differently (visited, unvisited)
* Style an element when it gets focus (focus)

The syntax of pseudo-classes:

|  |
| --- |
| selector:pseudo-class {  property:value;  } |

1. **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.

1. **Anchor Pseudo-classes**

The :first-child pseudo-class matches a specified element that is the first child of another element. In the following example, the selector matches any <p> element that is the first child of any element:

|  |
| --- |
| p:first-child {  color: blue;  } |

In the following example, the selector matches the first <i> element in all <p> elements:

|  |
| --- |
| p i:first-child {  color: blue;  }  <body>  <p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>  <p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>  </body> |

For a complete list of CSS pseudo classes, look at: <https://www.w3schools.com/css/css_pseudo_classes.asp>

**CSS 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

The syntax of pseudo-elements:

|  |
| --- |
| selector::pseudo-element {  property:value;  } |

1. **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:

|  |
| --- |
| p::first-line {  color: #ff0000;  font-variant: small-caps;  } |

1. **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:

|  |
| --- |
| p::first-letter {  color: #ff0000;  font-size: xx-large;  } |

1. **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:

|  |
| --- |
| h1::before {  content: url(smiley.gif);  } |

1. **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:

|  |
| --- |
| h1::after {  content: url(smiley.gif);  } |

1. **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:

|  |
| --- |
| ::selection {  color: red;  background: yellow;  } |

**CSS Attribute Selectors**

It is possible to style HTML elements that have specific attributes or attribute values.

1. **CSS [attribute] Selector**

The [attribute] selector is used to select elements with a specified attribute. The following example selects all <a> elements with a target attribute:

|  |
| --- |
| a[target] {  background-color: yellow;  } |

1. **CSS [attribute="value"] Selector**

The [attribute="value"] selector is used to select elements with a specified attribute and value. The following example selects all <a> elements with a target="\_blank" attribute:

|  |
| --- |
| a[target="\_blank"] {  background-color: yellow;  } |

1. **CSS [attribute~="value"] Selector**

The [attribute~="value"] selector is used to select elements with an attribute value containing a specified word. The following example selects all elements with a title attribute that contains a space-separated list of words, one of which is "flower":

|  |
| --- |
| [title~="flower"] {  border: 5px solid yellow;  } |

1. **CSS [attribute|="value"] Selector**

The [attribute|="value"] selector is used to select elements with the specified attribute starting with the specified value. The value has to be a whole word, either alone, like class="top", or followed by a hyphen( - ), like class="top-text". The following example selects all elements with a class attribute value that begins with "top":

|  |
| --- |
| [class|="top"] {  background: yellow;  } |

1. **CSS [attribute^="value"] Selector**

The [attribute^="value"] selector is used to select elements whose attribute value begins with a specified value. The following example selects all elements with a class attribute value that begins with "top":

|  |
| --- |
| [class^="top"] {  background: yellow;  } |

1. **CSS [attribute$="value"] Selector**

The [attribute$="value"] selector is used to select elements whose attribute value ends with a specified value. The following example selects all elements with a class attribute value that that ends with “test”.

|  |
| --- |
| [class$="test"] {  background: yellow;  } |

1. **CSS [attribute\*="value"] Selector**

The [attribute\*="value"] selector is used to select elements whose attribute value contains a specified value. The following example selects all elements with a class attribute value that contains "te":

|  |
| --- |
| [class\*="te"] {  background: yellow;  } |

**Summary of CSS Attribute Selectors**

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Example description** |
| [attribute] | [target] | Selects all elements with a target attribute |
| [attribute=value] | [target=\_blank] | Selects all elements with target="\_blank" |
| [attribute~=value] | [title~=flower] | Selects all elements with a title attribute containing the word "flower" |
| [attribute|=value] | [lang|=en] | Selects all elements with a lang attribute value starting with "en" |
| [attribute^=value] | a[href^="https"] | Selects every <a> element whose href attribute value begins with "https" |
| [attribute$=value] | a[href$=".pdf"] | Selects every <a> element whose href attribute value ends with ".pdf" |
| [attribute\*=value] | a[href\*="w3schools"] | Selects every <a> element whose href attribute value contains the substring "w3schools" |

**Summary of CSS Selectors**

There are many more and advanced CSS selectors. Here are some of the most widely used ones:

|  |
| --- |
|  |
|  |
|  |
|  |

The following Web site illustrates different CSS selectors using a game: <http://flukeout.github.io/>

Also look at the following Web site for further reference: <https://guide.freecodecamp.org/css/tutorials/css-selectors-cheat-sheet/>

**How different styles are applied: Cascading Order**

What style will be used when there is more than one style specified for an HTML element? All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults. When an element belongs to say multiple classes, and each class defines different values for the same CSS property, then the lastly defined property will be used. Here is an example:

|  |
| --- |
| <p class="class1 class2">This paragraph belongs to two classes.</p> |

Consider the above paragraph where <p> belongs to two classes “class1” and “class2”. What if both classes override the same property, e.g., color? Which one will take effect? The CSS rules in a CSS file are processed from top to bottom. In this case, the class that comes later **in the CSS file** will have the precedence, i.e., class2.

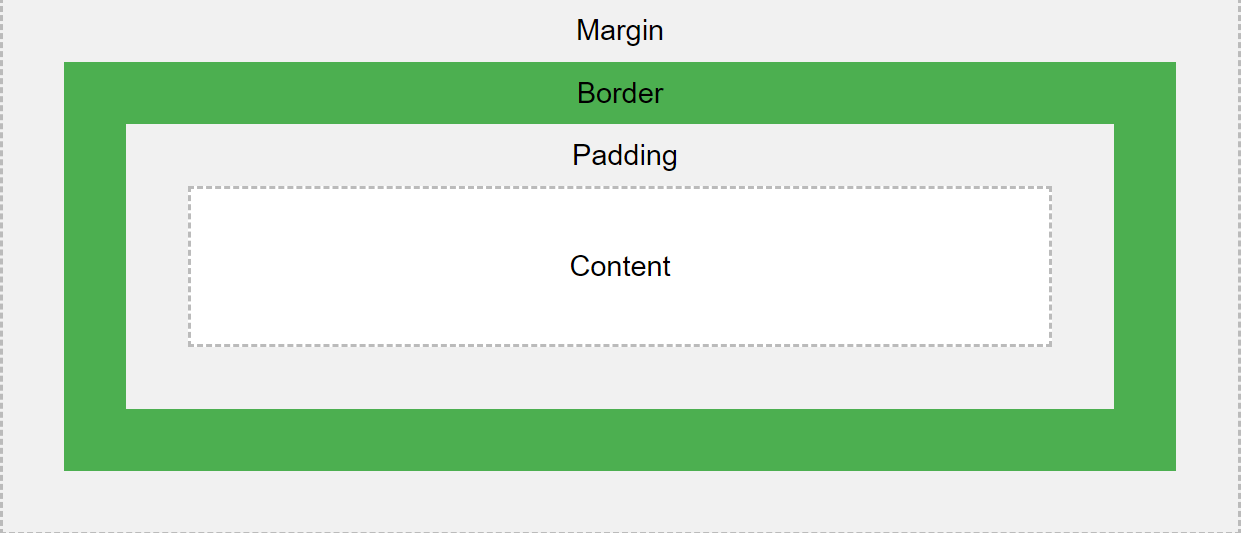
|  |
| --- |
| .class1 {  text-align: center;  color: green;  }  .class2 {  color: red;  font-size: 2em; /\* 2x16=32px \*/  } |

Look at: 02-CSS/05-Precedence

**Margins, Borders, and Padding**

All HTML elements can be considered as boxes. 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, where:

* **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



It is also possible to set the **width** & **height** of an HTML element. The height and width properties do not include padding, borders, or margins. It sets the height/width of the area inside the padding, border, and margin of the element. They may have the following values:

* auto - This is default. The browser calculates the height and width
* length - Defines the height/width in px, cm, “em” etc.
* % - Defines the height/width in percent of the containing block
* initial - Sets the height/width to its default value
* inherit - The height/width will be inherited from its parent value

The **max-width** property is used to set the maximum width of an element. The max-width can be specified in length values, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width).

Here is an example style for a div element:

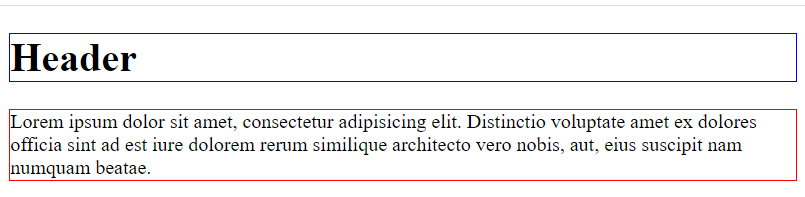
|  |
| --- |
| div {  width: 320px;  border: 5px solid red;  padding: 10px;  margin: 0px;  } |

With the above setup, the width of the div element becomes: 320px (width) + 20px (left + right padding) + 10px (left + right border) + 0px (left + right margin) = 350px. The height can be computed similarly.

Look at: 02-CSS/06-BoxModel

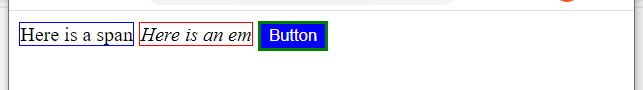
**Display: block, inline, inline-block, table, none**

Every HTML element has a default display style. For example, the default display property of h1, p, ul, div, section etc. is “block” by default. This means that when they are displayed within another “display: block” container such as “body”, the browser will move the cursor to the next line and give this element the entire width of the container. Then the cursor will move to the next line. For example, let’s say that we have the following web page with an h1 and a paragraph. As you can see both are placed on top of each other and occupy the entire width of the container.



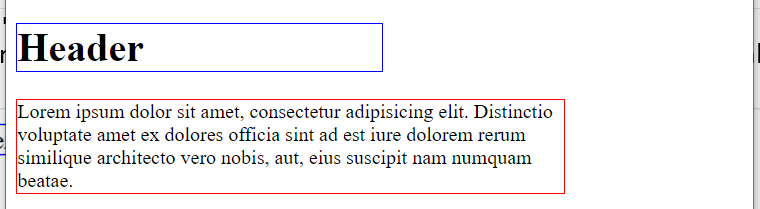
Look at: 02-CSS/07-Display/index1.html

Inline and inline-block elements such as span, em, button, etc. are placed side-by-side within their container and occupy only the enough space required by the element. Here is an example where we have three inline elements.



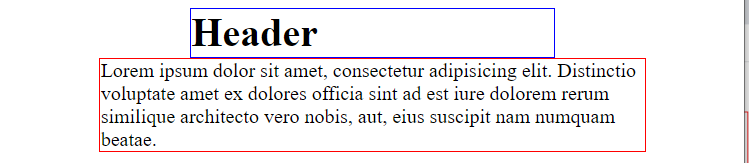
Look at: 02-CSS/07-Display/index2.html

If you want the block elements NOT to occupy the entire width of its container, you must specify its “width” property to a certain size. By default, the width of a block element is 100%. You can either set to a certain fixed size like “width: 200px”, or a certain percentage of the container’s width such as “width: 50%”. Here is how our block elements would look like if we set their width to 50% and 75% respectively:

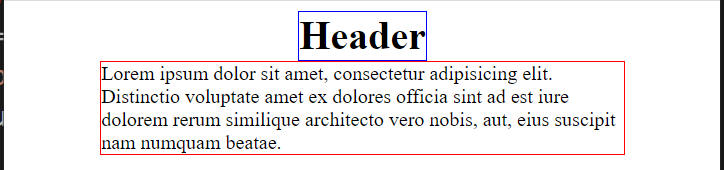


Look at: 02-CSS/07-Display/index3.html

Sometimes you want to vertically center these block containers. You can do that by specifying width: auto as in the following example: 02-CSS/07-Display/index4.html. As you can see, margin: auto only auto-adjusts the left and right margins. It has no effect on the top and bottom margins, which are both set to 0 now!

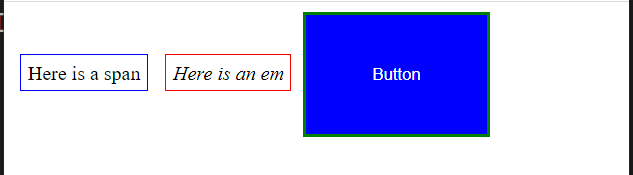


In the previous two examples, we see that although the header element requires a smaller amount of width, we had to specify a larger width (50% of its parent container) in order to make its width to be smaller than 100%. Sometimes, you want a block element’s width to be equal to what the element actually requires as in inline & inline-block elements. In that case, you need to set the display property of that element to be “display: table”. In the following example, we set the header’s display to be table and center it using margin: auto. Here is the result:



Look at: 02-CSS/07-Display/index5.html. We should note that if you set “display: table” for an element that occupies more than one line as is the case for our “p” element above, then it behave like a block element and occupies the entire width of the parent container unless you make it smaller than 100% by setting its width property to a smaller value. That is, when the contents of an element occupies more than 1 line then display: table and display: block work exactly the same.

We can also set the width & height of **inline-block** elements, but NOT **inline** elements. This is in fact the major difference between them. Even if you set the width & height properties of inline elements, the browser simply ignores them. Furthermore, you can NOT specify top & bottom margins for inline elements, while you CAN for inline-block elements (<https://www.w3schools.com/css/css_inline-block.asp>).



Look at: 02-CSS/07-Display/index6.html

**Text Formatting**

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 **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 **text-decoration** property is used to set or remove decorations from text. The value **text-decoration: none**; is often used to remove underlines from links. The other text-decoration values to decorate text are: *overline, underline, line-through*.

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. Possible values are *lowercase*, *uppercase*, *capitalize*.

The **text-indent** property is used to specify the indentation of the first line of a text.

The **font** properties define the *font family*, *boldness*, *size*, and the *style* of a text. There are two types of font family names:

* **generic family** - a group of font families with a similar look (like "Serif" or "Monospace")
* **font family** - a specific font family (like "Times New Roman" or "Arial")

The **font-family** property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on. For example:

|  |
| --- |
| p {  font-family: "Times New Roman", Times, serif;  } |

The **font-style** property is mostly used to specify italic text. Possible values are: *normal, italic, oblique*

The **font-size** property sets the size of the text. It is customary to specify font-sizes using “em”. 1em is equal to the current font size. The default text size in browsers is 16px. So, the default size of “1em” is 16px.

Look at: 02-CSS/08-TextFormatting

**Styling Lists**

The CSS list properties allow you to:

* Set different list item markers for ordered lists
* Set different list item markers for unordered lists
* Set an image as the list item marker
* Add background colors to lists and list items

The **list-style-type** property specifies the type of list item marker. Possible values are: *none*, *circle, square*, etc.

The **list-style-position** property specifies the position of the list-item markers. Possible values are: *inside, outside*

The **list-style-image** property specifies an image as the list item marker.

Look at: 02-CSS/09-Lists

**Styling Tables**

The look of an HTML table can be greatly improved with CSS.

To specify table borders in CSS, use the **border** property.

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

The table styled this way will have double borders. This is because both the table and the <th> and <td> elements have separate borders.

The **border-collapse** property sets whether the table borders should be collapsed into a single border:

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

Width and height of a table are defined by the **width** and **height** properties.

The **text-align** property sets the horizontal alignment (like *left, right,* or *center*) of the content in <th> or <td>.

The **vertical-align** property sets the vertical alignment (like *top, bottom,* or *middle*) of the content in <th> or <td>.

To control the space between the border and the content in a table, use the **padding** property on <td> and <th> elements.

Add the **border-bottom** property to <th> and <td> for horizontal dividers.

Use the **:hover** selector on <tr> to highlight table rows on mouse over:

|  |
| --- |
| tr:hover {background-color: #baecea;} |

For zebra-striped tables, use the nth-child() selector and add a background-color to all even (or odd) table rows:

|  |
| --- |
| tr:nth-child(even) {background-color: #f2f2f2;} |

A responsive table will display a horizontal scroll bar if the screen is too small to display the full content. Add a container element (like <div>) with overflow-x:auto around the <table> element to make it responsive.

Look at: 02-CSS/10-Tables

**Styling Forms**

The look of an HTML form can be greatly improved with CSS.

Use the **width** property to determine the width of the input field:

|  |
| --- |
| input {  width: 100%;  } |

The example above applies to all <input> elements. If you only want to style a specific input type, you can use attribute selectors:

* input[type=text] - will only select text fields
* input[type=password] - will only select password fields
* input[type=number] - will only select number fields

Setting **box-sizing** property to *border-box* makes sure that the padding and eventually borders are included in the total width and height of the elements.

By default, some browsers will add a blue outline around the input when it gets focus (clicked on). You can remove this behavior by adding outline: none; to the input. Use the **:focus** selector to do something with the input field when it gets focus:

|  |
| --- |
| input[type=text]:focus {  background-color: lightblue;  } |

If you want an icon inside the input, use the **background-image** property and position it with the **background-position** property. Also notice that we add a large left padding to reserve the space of the icon:

|  |
| --- |
| input[type=text] {  background-color: white;  background-image: url('searchicon.png');  background-position: 10px 10px;  background-repeat: no-repeat;  padding-left: 40px;  } |

Look at: 02-CSS/11-Forms

**Navigation Bars**

A navigation bar is basically a list of links, so using the <ul> and <li> elements makes perfect sense:

|  |
| --- |
| <ul>  <li><a href="default.asp">Home</a></li>  <li><a href="news.asp">News</a></li>  <li><a href="contact.asp">Contact</a></li>  <li><a href="about.asp">About</a></li>  </ul> |

With CSS you can transform boring HTML menus into good-looking navigation bars.

We can remove the bullets and the margins and padding from the list as follows:

|  |
| --- |
| ul {  list-style-type: none;  margin: 0;  padding: 0;  } |

**Vertical Navigation Bar**

To build a *vertical navigation bar*, you can style the <a> elements inside the list as follows:

|  |
| --- |
| li a {  display: block; /\* Makes the whole area clickable (not just the text) \*/  width: 60px;  } |

**display: block**; - Displaying the links as block elements makes the whole link area clickable (not just the text), and it allows us to specify the width (and padding, margin, height, etc. if you want)

**width: 60px**; - Block elements take up the full width available by default. We want to specify a 60 pixels width

**Horizontal Navigation Bar**

There are two ways to create a horizontal navigation bar. Using **inline** or **floating** list items.

|  |
| --- |
| li {  display: inline; /\* OR display: float; \*/  } |

**display: inline**; - By default, <li> elements are block elements. Here, we remove the line breaks before and after each list item, to display them on one line. **display: float**; would do the same.

Look at: 02-CSS/12-NavBars

**CSS Variables (a.k.a. CSS Custom Properties)**

We have seen so far that you need to write a CSS style file and then use your style file to style your web page. Notice that the CSS style file has many hard-coded constants. Let’s say that in the future you want to change some of these constants to other values. Then you need to go through your entire CSS file, find the appropriate places that you need to change and then change them, which may be pain in the neck. This is similar to putting all your constant values directly in your code, and then going and changing them. As your recall from your programming courses, you are advised to define these constants (PI, E, …) at the top of your program and use the symbolic constants instead. If, for some reason, you want to change these values, you can simply update their value at one place, and it will be reflected to all places in your code by the pre-processor.

The same problem applies to CSS files. To solve this problem, CSS variables have been invented. At the simplest level, you define a CSS variable at the top and then use it in your CSS rules as follows:

|  |
| --- |
| :root {  --main-bg-color: #f4f4f4  }  .box {  background-color: var(--main-bg-color)  } |

Look at: 02-CSS/13-CSS-Variables/index1.html

For a more detailed example look at the example by Brad Traversy: 02-CSS/13-CSS-Variables/index2.html

This code was taken from Traversy’s CSS Variables tutorial: <https://www.youtube.com/watch?v=sQUB039MG0I>

**CSS Preprocessors/Compilers**

Before CSS variables were invented and have become part of the CSS standard, people invented CSS Preprocessors/Compilers for the same purpose. The most popular CSS preprocessor/compiler is the Syntactically Awesome StyleSheets (SaSS) (<https://sass-lang.com/>), which lets not only lets you write CSS rules with variables, but it is more like a PL letting you write CSS rules with “if” statements, “loops”, and “functions with parameters”. The SaSS preprocessor/compiler then converts this file into a CSS file that you can use in your HTML files. The most popular CSS preprocessor for SaSS is gulp (<https://gulpjs.com/>). Install node and run the following command to create a node project and install gulp & associated modules. They are installed using npm as follows:

% npm init

% npm install gulp gulp-sass sass –save-dev

Look at: 02-CSS/14-SaSS

**More Resources to learn HTML, CSS & SaSS**

Here are a couple of videos where you can learn more about the HTML, CSS & SaSS:

1. Net Ninja (HTML & CSS): <https://www.youtube.com/playlist?list=PL4cUxeGkcC9ivBf_eKCPIAYXWzLlPAm6G>
2. Traversy (CSS): <https://www.youtube.com/watch?v=yfoY53QXEnI&list=RDCMUC29ju8bIPH5as8OGnQzwJyA>
3. Net Ninja (CSS variables): <https://www.youtube.com/playlist?list=PL4cUxeGkcC9jXaLsxbEmsPSOlb40ZLaKN>
4. Brad Travery (CSS variables): <https://www.youtube.com/watch?v=sQUB039MG0I>
5. Net Ninja (SaSS Tutorial): <https://www.youtube.com/playlist?list=PL4cUxeGkcC9jxJX7vojNVK-o8ubDZEcNb>
6. Brad Traversy (SaSS crash course): <https://www.youtube.com/watch?v=nu5mdN2JIwM>