CSS Basics

Teacher: Ms. Ahmad Osman

Date: 04/01/2019

Overview & Purpose

In this tutorial, the participants will learn the skills that he needs in CSS to build a complete responsive webpage

Education Standards

The participant should have knowledge about HTML5 like Tags, classes, attributes

Objectives

1. Create CSS files
2. CSS attributes and selectors.
3. Responsive design.
4. Write Clean and maintainable code
5. CSS techniques

Software Needed

1. Visual Studio code and codepen

Verification

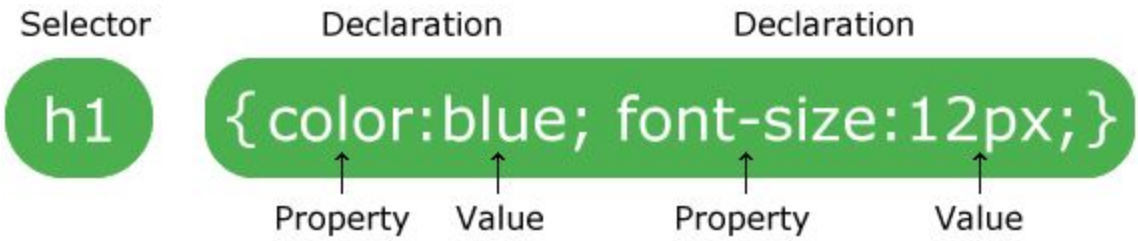
1. Using Selectors.
2. Create Responsive webpage.
3. Use CSS libraries

Activity

Learning using coding and exercises, discover the CSS and how to use it.

CSS Syntax

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



Example

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

CSS Selectors
The element Selector

The element selector selects elements based on the element name.
You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color like the example above.

The id Selector

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

The id of an element should be 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 style rule below will be applied to the HTML element with id="para1":

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

The class Selector

The class selector selects elements with a specific class attribute.

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

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

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

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

In the example below, only <p> elements with class="center" will be center-aligned:

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

HTML elements can also refer to more than one class.

In the example below, 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>
```

Grouping Selectors

If you have elements with the same style definitions, like this:

```
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.

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

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

CSS Comments

Comments are used to explain the code and may help when you edit the source code at a later date.

Comments are ignored by browsers.

A CSS comment starts with /* and ends with */. Comments can also span multiple lines:

```
p {
  color: red;
  /* This is a single-line comment */
  text-align: center;
}

/* This is
a multi-line
comment */
```

CSS Colors

Color Names

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

```
<h1 style="background-color:Tomato;">Tomato</h1>
<h1 style="background-color:Orange;">Orange</h1>
<h1 style="background-color:DodgerBlue;">DodgerBlue</h1>
<h1 style="background-color:MediumSeaGreen;">MediumSeaGreen</h1>
<h1 style="background-color:Gray;">Gray</h1>
<h1 style="background-color:SlateBlue;">SlateBlue</h1>
<h1 style="background-color:Violet;">Violet</h1>
<h1 style="background-color:LightGray;">LightGray</h1>
```

Background Color

You can set the background color for HTML elements:

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

Text Color

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

Border Color

```
<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:

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Same as color name "Tomato", but 50% transparent:

rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

```
<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>
```

RGB Value

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 between 0 and 255.

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

To display the color black, all color parameters must be set to 0, like this: RGB(0, 0, 0).

To display the color white, all color parameters must be set to 255, like this: RGB(255, 255, 255).

Experiment by mixing the RGB values below:

RGB(255, 99, 71)

HEX Value

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 others are set to the lowest value(00).

HSL Value

n 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, 0% is black, 50% is neither light or dark, 100% is white

RGBA Value

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)

HSLA Value

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)

CSS Backgrounds

Background Color

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

The background color of a page is set like this:

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

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.

The background image for a page can be set like this:

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

Background Image - Repeat Horizontally or Vertically

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

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

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

Showing the background image only once is also specified by the background-repeat: no-repeat;

Background-position

The position of the image is specified by the background-position property:

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

Background Image - Fixed position

To specify that the background image should be fixed (will not scroll with the rest of the page), use the background-attachment property:

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

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.

The shorthand property for the background is background:

```
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

CSS Borders

The CSS border properties allow you to specify the style, width, and color of an element's border.

Border Style

The border-style property specifies what kind of border to display.

The following values are allowed:

dotted - Defines a dotted border
dashed - Defines a dashed border
solid - Defines a solid border
double - Defines a double border
groove - Defines a 3D grooved border. The effect depends on the border-color value
ridge - Defines a 3D ridged border. The effect depends on the border-color value
inset - Defines a 3D inset border. The effect depends on the border-color value
outset - Defines a 3D outset border. The effect depends on the border-color value
none - Defines no border
hidden - Defines a hidden border
The border-style property can have from one to four values (for the top border, right border, bottom border, and the left border).

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
```

Border Width

The border-width property specifies the width of the four borders.

The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick.

The border-width property can have from one to four values (for the top border, right border, bottom border, and the left border).

```
p.one {
  border-style: solid;
  border-width: 5px;
}

p.two {
  border-style: solid;
  border-width: medium;
}

p.three {
  border-style: solid;
  border-width: 2px 10px 4px 20px;
}
```

Border Color

The border-color property is used to set the color of the four borders.

The color can be set by:

name - specify a color name, like "red"
Hex - specify a hex value, like "#ff0000"
RGB - specify a RGB value, like "rgb(255,0,0)"
transparent
The border-color property can have from one to four values (for the top border, right border, bottom border, and the left border).

If border-color is not set, it inherits the color of the element.

```
p.one {
  border-style: solid;
  border-color: red;
}

p.two {
  border-style: solid;
  border-color: green;
}

p.three {
  border-style: solid;
  border-color: red green blue yellow;
}
```

Border - Individual Sides

From the examples above you have seen that it is possible to specify a different border for each side.

In CSS, there are also properties for specifying each of the borders (top, right, bottom, and left):

```
p {
  border-top-style: dotted;
  border-right-style: solid;
  border-bottom-style: dotted;
  border-left-style: solid;
}
```

Border - Shorthand Property

As you can see from the examples above, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the individual border properties in one property.

The border property is a shorthand property for the following individual border properties:

border-width
border-style (required)
Border-color

```
p {
  border: 5px solid red;
}
```

Rounded Borders

The border-radius property is used to add rounded borders to an element:

```
p {
  border: 2px solid red;
  border-radius: 5px;
}
```

Margins

The CSS `margin` properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

CSS has properties for specifying the margin for each side of an element:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

All the margin properties can have the following values:

- auto - the browser calculates the margin
- *length* - specifies a margin in px, pt, cm, etc.

- % - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element

Tip: Negative values are allowed.

```
p {
      margin-top: 100px;
      margin-bottom: 100px;
      margin-right: 150px;
      margin-left: 80px;
}
```

Margin - Shorthand Property

To shorten the code, it is possible to specify all the margin properties in one property.

The `margin` property is a shorthand property for the following individual margin properties:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

So, here is how it works:

If the `margin` property has four values:

- **margin: 25px 50px 75px 100px;**
  - top margin is 25px
  - right margin is 50px
  - bottom margin is 75px
  - left margin is 100px

```
p {

  margin: 25px 50px 75px 100px;

}
```

If the `margin` property has three values:

- **margin: 25px 50px 75px;**
  - top margin is 25px
  - right and left margins are 50px
  - bottom margin is 75px

```
p {
  margin: 25px 50px 75px;
}
```

If the `margin` property has two values:

- **margin: 25px 50px;**
  - top and bottom margins are 25px
  - right and left margins are 50px

```
p {
  margin: 25px 50px;
}
```

If the `margin` property has one value:

- **margin: 25px;**
  - all four margins are 25px

```
p {
  margin: 25px;
}
```

The auto Value

You can set the margin property to `auto` to horizontally center the element within its container.
The element will then take up the specified width, and the remaining space will be split equally between the left and right margins:

```
div {
  width: 300px;
  margin: auto;
  border: 1px solid red;
}
```

The inherit Value

This example lets the left margin of the <p class="ex1"> element be inherited from the parent element ( ):

```
div {
  border: 1px solid red;
  margin-left: 100px;
}

p.ex1 {
  margin-left: inherit;
```

```
}
```

Margin Collapse

Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.
This does not happen on left and right margins! Only top and bottom margins!
Look at the following example:

```
h1 {

  margin: 0 0 50px 0;

}



h2 {

  margin: 20px 0 0 0;

}
```

Padding

The CSS `padding` properties are used to generate space around an element's content, inside of any defined borders.
With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

All the padding properties can have the following values:

- *length* - specifies a padding in px, pt, cm, etc.

- % - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element

Note: Negative values are not allowed.
The following example sets different padding for all four sides of a element:

```
div {
  padding-top: 50px;
  padding-right: 30px;
  padding-bottom: 50px;
  padding-left: 80px;
}
```

Padding - Shorthand Property

To shorten the code, it is possible to specify all the padding properties in one property.
The `padding` property is a shorthand property for the following individual padding properties:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

So, here is how it works:
If the `padding` property has four values:

- **padding: 25px 50px 75px 100px;**
  - top padding is 25px
  - right padding is 50px
  - bottom padding is 75px
  - left padding is 100px

```
div {

  padding: 25px 50px 75px 100px;

}
```

If the `padding` property has three values:

- **padding: 25px 50px 75px;**
  - top padding is 25px
  - right and left paddings are 50px

- bottom padding is 75px

```
div {

  padding: 25px 50px 75px;

}
```

If the `padding` property has two values:

- **padding: 25px 50px;**
  - top and bottom paddings are 25px
  - right and left paddings are 50px

```
div {

  padding: 25px 50px;

}
```

If the `padding` property has one value:

- **padding: 25px;**
  - all four paddings are 25px

```
div {

  padding: 25px;

}
```

Padding and Element Width

The CSS `width` property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element
So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.
In the following example, the element is given a width of 300px. However, the actual rendered width of the element will be 350px (300px + 25px of left padding + 25px of right padding):

```
div {

  width: 300px;

  padding: 25px;

}
```

To keep the width at 300px, no matter the amount of padding, you can use the `box-sizing` property. This causes the element to maintain its width; if you increase the padding, the available content space will decrease. Here is an example:

```
div {
  width: 300px;
  padding: 25px;
  box-sizing: border-box;
}
```

Height and Width

The `height` and `width` properties are used to set the height and width of an element.
The `height` and `width` can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in *length values*, like px, cm, etc., or in percent (%) of the containing block.

```
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}

div {
  height: 100px;
  width: 500px;
  background-color: powderblue;
}
```

Note: The `height` and `width` properties do not include padding, borders, or margins; they set the height/width of the area inside the padding, border, and margin of the element!

Setting max-width

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).
The problem with the `<div>` above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page.
Using `max-width` instead, in this situation, will improve the browser's handling of small windows.
Tip: Drag the browser window to smaller than 500px wide, to see the difference between the two divs!
Note: The value of the `max-width` property overrides `width`.
The following example shows a `<div>` element with a height of 100 pixels and a max-width of 500 pixels:

```
div {
  max-width: 500px;
  height: 100px;
  background-color: powderblue;
}
```

CSS Outline

An outline is a line that is drawn around elements, OUTSIDE the borders, to make the element "stand out".
CSS has the following outline properties:

- `outline-style`
- `outline-color`
- `outline-width`
- `outline-offset`
- `outline`

Outline Style

The `outline-style` property specifies the style of the outline, and can have one of the following values:

- `dotted` - Defines a dotted outline
- `dashed` - Defines a dashed outline
- `solid` - Defines a solid outline
- `double` - Defines a double outline
- `groove` - Defines a 3D grooved outline
- `ridge` - Defines a 3D ridged outline
- `inset` - Defines a 3D inset outline
- `outset` - Defines a 3D outset outline
- `none` - Defines no outline
- `hidden` - Defines a hidden outline

```
p.dotted {outline-style: dotted;}
p.dashed {outline-style: dashed;}
p.solid {outline-style: solid;}
p.double {outline-style: double;}
p.groove {outline-style: groove;}
p.ridge {outline-style: ridge;}
p.inset {outline-style: inset;}
p.outset {outline-style: outset;}
```

Outline Color

The `outline-color` property is used to set the color of the outline.
The color can be set by:

- name - specify a color name, like "red"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- Hex - specify a hex value, like "#ff0000"
- invert - performs a color inversion (which ensures that the outline is visible, regardless of color background)

The following example shows some different outlines with different colors. Also notice that these elements also have a thin black border inside the outline:

```
p.ex1 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
```

```
}

p.ex2 {
  border: 1px solid black;
  outline-style: double;
  outline-color: green;
}

p.ex3 {
  border: 1px solid black;
  outline-style: outset;
  outline-color: yellow;
}
```

The following example uses `outline-color: invert`, which performs a color inversion. This ensures that the outline is visible, regardless of color background:

```
p.ex1 {
  border: 1px solid yellow;
  outline-style: solid;
  outline-color: invert;
}
```

Outline Width

The `outline-width` property specifies the width of the outline, and can have one of the following values:

- thin (typically 1px)
- medium (typically 3px)
- thick (typically 5px)
- A specific size (in px, pt, cm, em, etc)

The following example shows some outlines with different widths:

```
p.ex1 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
  outline-width: thin;
}

p.ex2 {
  border: 1px solid black;
```

```
  outline-style: solid;
  outline-color: red;
  outline-width: medium;
}

p.ex3 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
  outline-width: thick;
}

p.ex4 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
  outline-width: 4px;
}
```

Outline - Shorthand property

The `outline` property is a shorthand property for setting the following individual outline properties:

- `outline-width`
- `outline-style` (required)
- `outline-color`

The `outline` property is specified as one, two, or three values from the list above. The order of the values does not matter.
The following example shows some outlines specified with the shorthand `outline` property:

```
p.ex1 {outline: dashed;}
p.ex2 {outline: dotted red;}
p.ex3 {outline: 5px solid yellow;}
p.ex4 {outline: thick ridge pink;}
```

Outline Offset

The `outline-offset` property adds space between an outline and the edge/border of an element. The space between an element and its outline is transparent.

The following example specifies an outline 15px outside the border edge:

```
p {
  margin: 30px;
  border: 1px solid black;
  outline: 1px solid red;
  outline-offset: 15px;
}
```

The following example shows that the space between an element and its outline is transparent:

```
p {
  margin: 30px;
  background: yellow;
  border: 1px solid black;
  outline: 1px solid red;
  outline-offset: 15px;
}
```

CSS Text

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 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):

```
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 Decoration

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:

```
a {
  text-decoration: none;
}
```

The other `text-decoration` values are used to decorate text:

```
h1 {
  text-decoration: overline;
}

h2 {
  text-decoration: line-through;
}

h3 {
  text-decoration: underline;
```

```
}
```

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:

```
p.uppercase {
  text-transform: uppercase;
}

p.lowercase {
  text-transform: lowercase;
}

p.capitalize {
  text-transform: capitalize;
}
```

Text Indentation

The `text-indent` property is used to specify the indentation of the first line of a text:

```
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:

```
h1 {
  letter-spacing: 3px;
}

h2 {
```

```
  letter-spacing: -3px;
}
```

Line Height

The `line-height` property is used to specify the space between lines:

```
p.small {
  line-height: 0.8;
}

p.big {
  line-height: 1.8;
}
```

Text Direction

The `direction` property is used to change the text direction of an element:

```
p {
  direction: rtl;
}
```

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:

```
h1 {
  word-spacing: 10px;
}

h2 {
  word-spacing: -5px;
}
```

Text Shadow

The `text-shadow` property adds shadow to text.
The following example specifies the position of the horizontal shadow (3px), the position of the vertical shadow (2px) and the color of the shadow (red):

```
h1 {
  text-shadow: 3px 2px red;
```

```
}
```

CSS Fonts

The CSS font properties define the font family, boldness, size, and the style of a text.

CSS Font Families

In CSS, 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")

Font Family

The font family of a text is set with the `font-family` property.
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.
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.
Note: If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".
More than one font family is specified in a comma-separated list:

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

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)

```
p.normal {
  font-style: normal;
}
```

```
p.italic {
  font-style: italic;
}

p.oblique {
  font-style: oblique;
}
```

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

Set Font Size With Pixels

Setting the text size with pixels gives you full control over the text size:

```
h1 {
  font-size: 40px;
}

h2 {
  font-size: 30px;
}

p {
  font-size: 14px;
```

```
}
```

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*

```
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 IE. 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:

```
body {
  font-size: 100%;
}

h1 {
  font-size: 2.5em;
}

h2 {
  font-size: 1.875em;
}

p {
  font-size: 0.875em;
}
```

Font Weight

The `font-weight` property specifies the weight of a font:

```
p.normal {
```

```
  font-weight: normal;

}


p.thick {

  font-weight: bold;

}
```

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>
```

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.

```
p.normal {
  font-variant: normal;
}

p.small {
  font-variant: small-caps;
}
```

CSS Icons

How To Add Icons

The simplest way to add an icon to your HTML page, is with an icon library, such as Font Awesome.
Add the name of the specified icon class to any inline HTML element (like `<i>` or `<span>`).
All the icons in the icon libraries below, are scalable vectors that can be customized with CSS (size, color, shadow, etc.)

Font Awesome Icons

To use the Font Awesome icons, add the following line inside the `<head>` section of your HTML page:

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/
font-awesome.min.css">
```

Note: No downloading or installation is required!

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/fo
nt-awesome.min.css">
</head>
<body>

<i class="fa fa-cloud"></i>
<i class="fa fa-heart"></i>
<i class="fa fa-car"></i>
<i class="fa fa-file"></i>
<i class="fa fa-bars"></i>

</body>
</html>
```

Bootstrap Icons

To use the Bootstrap glyphicons, add the following line inside the `<head>` section of your HTML page:

`<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">`

Note: No downloading or installation is required!

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
</head>
<body>

<i class="glyphicon glyphicon-cloud"></i>
<i class="glyphicon glyphicon-remove"></i>
<i class="glyphicon glyphicon-user"></i>
<i class="glyphicon glyphicon-envelope"></i>
<i class="glyphicon glyphicon-thumbs-up"></i>

</body>
</html>
```

Google Icons

To use the Google icons, add the following line inside the `<head>` section of your HTML page:

`<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">`

Note: No downloading or installation is required!

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
</head>
<body>

<i class="material-icons">cloud</i>
<i class="material-icons">favorite</i>
<i class="material-icons">attachment</i>
```

```
<i class="material-icons">computer</i>
<i class="material-icons">traffic</i>

</body>
</html>
```

CSS Links

Styling Links

Links can be styled with any CSS property (e.g. `color`, `font-family`, `background`, etc.).

```
a {
  color: hotpink;
}
```

In addition, links can be styled differently depending on what **state** they are in.
The four links states are:

- `a:link` - a normal, unvisited link
- `a:visited` - a link the user has visited
- `a:hover` - a link when the user mouses over it
- `a:active` - a link the moment it is clicked

```
/* unvisited link */
a:link {
  color: red;
}

/* visited link */
a:visited {
  color: green;
}

/* mouse over link */
a:hover {
  color: hotpink;
}

/* selected link */
a:active {
  color: blue;
```

```
}
```

When setting the style for several link states, there are some order rules:

- a:hover MUST come after a:link and a:visited
- a:active MUST come after a:hover

Text Decoration

The `text-decoration` property is mostly used to remove underlines from links:

```
a:link {
  text-decoration: none;
}

a:visited {
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

a:active {
  text-decoration: underline;
}
```

Background Color

The `background-color` property can be used to specify a background color for links:

```
a:link {
  background-color: yellow;
}

a:visited {
  background-color: cyan;
}

a:hover {
  background-color: lightgreen;
}

a:active {
```

```
  background-color: hotpink;
}
```

Advanced - Link Buttons

This example demonstrates a more advanced example where we combine several CSS properties to display links as boxes/buttons:

```
a:link, a:visited {
  background-color: #f44336;
  color: white;
  padding: 14px 25px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
}

a:hover, a:active {
  background-color: red;
}
```

CSS Lists

HTML Lists and CSS List Properties

In HTML, there are two main types of lists:

- unordered lists (<ul>) - the list items are marked with bullets
- ordered lists (<ol>) - the list items are marked with numbers or letters

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

Different List Item Markers

The `list-style-type` property specifies the type of list item marker.
The following example shows some of the available list item markers:

```
ul.a {
  list-style-type: circle;
```

```
}

ul.b {
  list-style-type: square;
}

ol.c {
  list-style-type: upper-roman;
}

ol.d {
  list-style-type: lower-alpha;
}
```

An Image as The List Item Marker

The `list-style-image` property specifies an image as the list item marker:

```
ul {
  list-style-image: url('sqpurple.gif');
}
```

Position The List Item Markers

The `list-style-position` property specifies the position of the list-item markers (bullet points).
"list-style-position: outside;" means that the bullet points will be outside the list item. The start of each line of a list item will be aligned vertically. This is default:

```
ul.a {
  list-style-position: outside;
}
```

"list-style-position: inside;" means that the bullet points will be inside the list item. As it is part of the list item, it will be part of the text and push the text at the start:

```
ul.b {
  list-style-position: inside;
}
```

Remove Default Settings

The `list-style-type:none` property can also be used to remove the markers/bullets. Note that the list also has default margin and padding. To remove this, add `margin:0` and `padding:0` to <ul> or <ol>:

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

List - Shorthand property

The `list-style` property is a shorthand property. It is used to set all the list properties in one declaration:

```
ul {
  list-style: square inside url("sqpurple.gif");
}
```

When using the shorthand property, the order of the property values are:

- `list-style-type` (if a list-style-image is specified, the value of this property will be displayed if the image for some reason cannot be displayed)
- `list-style-position` (specifies whether the list-item markers should appear inside or outside the content flow)
- `list-style-image` (specifies an image as the list item marker)

If one of the property values above are missing, the default value for the missing property will be inserted, if any.

Styling List With Colors

We can also style lists with colors, to make them look a little more interesting. Anything added to the <ol> or <ul> tag, affects the entire list, while properties added to the <li> tag will affect the individual list items:

```
ol {
  background: #ff9999;
  padding: 20px;
}

ul {
```

```
  background: #3399ff;
  padding: 20px;
}

ol li {
  background: #ffe5e5;
  padding: 5px;
  margin-left: 35px;
}

ul li {
  background: #cce5ff;
  margin: 5px;
}
```

CSS Tables

Table Borders

To specify table borders in CSS, use the `border` property.
The example below specifies a black border for <table>, <th>, and <td> elements:

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

Collapse Table 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;
}
```

If you only want a border around the table, only specify the `border` property for <table>:

```
table {
  border: 1px solid black;
```

```
}
```

Table Width and Height

Width and height of a table are defined by the `width` and `height` properties.
The example below sets the width of the table to 100%, and the height of the <th> elements to 50px:

```
table {
  width: 100%;
}

th {
  height: 50px;
}
```

Horizontal Alignment

The `text-align` property sets the horizontal alignment (like left, right, or center) of the content in <th> or <td>.
By default, the content of <th> elements are center-aligned and the content of <td> elements are left-aligned.
The following example left-aligns the text in <th> elements:

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

Vertical Alignment

The `vertical-align` property sets the vertical alignment (like top, bottom, or middle) of the content in <th> or <td>.
By default, the vertical alignment of the content in a table is middle (for both <th> and <td> elements).
The following example sets the vertical text alignment to bottom for <td> elements:

```
td {
  height: 50px;
  vertical-align: bottom;
}
```

Table Padding

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

```
th, td {
  padding: 15px;
  text-align: left;
}
```

Horizontal Dividers

Add the `border-bottom` property to <th> and <td> for horizontal dividers:

```
th, td {
  border-bottom: 1px solid #ddd;
}
```

Hoverable Table

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

```
tr:hover {background-color: #f5f5f5;}
```

Striped Tables

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;}
```

Table Color

The example below specifies the background color and text color of <th> elements:

```
th {
  background-color: #4CAF50;
  color: white;
}
```

Responsive Table

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) with `overflow-x:auto` around the <table> element to make it responsive:

```
<div style="overflow-x:auto;">
```

```
<table>
... table content ...
</table>

</div>
```

CSS Layout - The display Property

The `display` property is the most important CSS property for controlling layout.

The display Property

The `display` property specifies if/how an element is displayed.
Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is `block` or `inline`.

Block-level Elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).
Examples of block-level elements:

-
- <h1> - <h6>
- <p>
- <form>
- <header>
- <footer>
- <section>

Inline Elements

An inline element does not start on a new line and only takes up as much width as necessary.
This is an inline element inside a paragraph.
Examples of inline elements:

-
- <a>
- <img>

Display: none;

`display: none;` is commonly used with JavaScript to hide and show elements without deleting and recreating them. Take a look at our last example on this page if you want to know how this can be achieved.
The `<script>` element uses `display: none;` as default.

Override The Default Display Value

As mentioned, every element has a default display value. However, you can override this.
Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.
A common example is making inline `<li>` elements for horizontal menus:

```
li {
  display: inline;
}
```

The following example displays elements as block elements:

```
span {
  display: block;
}
```

The following example displays <a> elements as block elements:

```
a {
  display: block;
}
```

Hide an Element - display:none or visibility:hidden?

Hiding an element can be done by setting the `display` property to `none`. The element will be hidden, and the page will be displayed as if the element is not there:

```
h1.hidden {
  display: none;
}
```

`visibility:hidden;` also hides an element.

```
h1.hidden {
  visibility: hidden;
}
```

The display: inline-block Value

Compared to `display: inline`, the major difference is that `display: inline-block` allows to set a width and height on the element.
Also, with `display: inline-block`, the top and bottom margins/paddings are respected, but with `display: inline` they are not.
Compared to `display: block`, the major difference is that `display: inline-block` does not add a line-break after the element, so the element can sit next to other elements.
The following example shows the different behavior of `display: inline`, `display: inline-block` and `display: block`:

```
<!DOCTYPE html>
<html>
<head>
<style>
span.a {
  display: inline; /* the default for span */
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}

span.b {
  display: inline-block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}

span.c {
  display: block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
```

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

<h1>The display Property</h1>

<h2>display: inline</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Vestibulum consequat scelerisque elit sit amet consequat. Aliquam
erat volutpat. <span class="a">Aliquam</span> <span
class="a">venenatis</span> gravida nisl sit amet facilisis. Nullam
cursus fermentum velit sed laoreet. </div>

<h2>display: inline-block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Vestibulum consequat scelerisque elit sit amet consequat. Aliquam
erat volutpat. <span class="b">Aliquam</span> <span
class="b">venenatis</span> gravida nisl sit amet facilisis. Nullam
cursus fermentum velit sed laoreet. </div>

<h2>display: block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Vestibulum consequat scelerisque elit sit amet consequat. Aliquam
erat volutpat. <span class="c">Aliquam</span> <span
class="c">venenatis</span> gravida nisl sit amet facilisis. Nullam
cursus fermentum velit sed laoreet. </div>

</body>
</html>
```

CSS Layout - width and max-width

Using width, max-width and margin: auto;

As mentioned in the previous chapter; a block-level element always takes up the full width available (stretches out to the left and right as far as it can). Setting the `width` of a block-level element will prevent it from stretching out to the edges of its container. Then, you can set the margins to auto, to

horizontally center the element within its container. The element will take up the specified width, and the remaining space will be split equally between the two margins:
Tip: Resize the browser window to less than 500px wide, to see the difference between the two divs!
Here is an example of the two divs above:

```
div.ex1 {
  width: 500px;
  margin: auto;
  border: 3px solid #73AD21;
}

div.ex2 {
  max-width: 500px;
  margin: auto;
  border: 3px solid #73AD21;
}
```

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:

```
div.static {
  position: static;
  border: 3px solid #73AD21;
}
```

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.

```
div.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}
```

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:

```
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
```

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: A "positioned" element is one whose position is anything except `static`.

```
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;
}
```

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).

```
div.sticky {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
  background-color: green;
  border: 2px solid #4CAF50;
}
```

Overlapping Elements

When elements are positioned, they can overlap other elements.
The `z-index` property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

An element can have a positive or negative stack order:

```
img {
  position: absolute;
  left: 0px;
  top: 0px;
  z-index: -1;
}
```

Positioning Text In an Image

How to position text over an image:

```
<!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>
```

CSS Layout - Overflow

The CSS `overflow` property controls what happens to content that is too big to fit into an area.
The `overflow` property specifies whether to clip content or to add scrollbars when the content of an element is too big to fit in a specified area.
The `overflow` property has the following values:

- `visible` - Default. The overflow is not clipped. It renders outside the element's box
- `hidden` - The overflow is clipped, and the rest of the content will be invisible
- `scroll` - The overflow is clipped, but a scrollbar is added to see the rest of the content
- `auto` - If overflow is clipped, a scrollbar should be added to see the rest of the content

Note: The `overflow` property only works for block elements with a specified height.
Note: In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).

overflow: visible

By default, the overflow is `visible`, meaning that it is not clipped and it renders outside the element's box:

```
div {
  width: 200px;
  height: 50px;
  background-color: #eee;
  overflow: visible;
}
```

overflow: hidden

With the `hidden` value, the overflow is clipped, and the rest of the content is hidden:

```
div {
  overflow: hidden;
}
```

overflow: scroll

Setting the value to `scroll`, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):

```
div {
  overflow: scroll;
}
```

overflow: auto

The `auto` value is similar to `scroll`, only it add scrollbars when necessary:

```
div {
  overflow: auto;
}
```

overflow-x and overflow-y

The `overflow-x` and `overflow-y` properties specifies whether to change the overflow of content just horizontally or vertically (or both):
`overflow-x` specifies what to do with the left/right edges of the content.
`overflow-y` specifies what to do with the top/bottom edges of the content.

```
div {
  overflow-x: hidden; /* Hide horizontal scrollbar */
  overflow-y: scroll; /* Add vertical scrollbar */
}
```

CSS Layout - float and clear

The CSS `float` property specifies how an element should float.
The CSS `clear` property specifies what elements can float beside the cleared element and on which side.

The float Property

The `float` property is used for positioning and formatting content e.g. let an image float left to the text in a container.
The `float` property can have one of the following values:

- left - The element floats to the left of its container
- right- The element floats to the right of its container
- none - The element does not float (will be displayed just where it occurs in the text). This is default
- inherit - The element inherits the float value of its parent

In its simplest use, the `float` property can be used to wrap text around images.

```
img {
  float: right;
}
img {
  float: left;
}
img {
  float: none;
}
```

The clear Property

The `clear` property specifies what elements can float beside the cleared element and on which side.
The `clear` property can have one of the following values:

- none - Allows floating elements on both sides. This is default
- left - No floating elements allowed on the left side
- right- No floating elements allowed on the right side
- both - No floating elements allowed on either the left or the right side
- inherit - The element inherits the clear value of its parent

The most common way to use the `clear` property is after you have used a `float` property on an element.
When clearing floats, you should match the clear to the float: If an element is floated to the left, then you should clear to the left. Your floated element will continue to float, but the cleared element will appear below it on the web page.
The following example clears the float to the left. Means that no floating elements are allowed on the left side (of the div):

```
div {
  clear: left;
```

```
}
```

The clearfix Hack

If an element is taller than the element containing it, and it is floated, it will "overflow" outside of its container:

```
.clearfix {
  overflow: auto;
}
```

The `overflow: auto` clearfix works well as long as you are able to keep control of your margins and padding (else you might see scrollbars). The **new, modern clearfix hack** however, is safer to use, and the following code is used for most webpages:

```
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}
```

Grid of Boxes / Equal Width Boxes

With the `float` property, it is easy to float boxes of content side by side:

```
* {
  box-sizing: border-box;
}

.box {
  float: left;
  width: 33.33%; /* three boxes (use 25% for four, and 50% for two,
etc) */
  padding: 50px; /* if you want space between the images */
}
```

CSS Layout - Horizontal & Vertical Align

Center Align Elements

To horizontally center a block element (like), use `margin: auto;`
Setting the width of the element will prevent it from stretching out to the edges of its container.

The element will then take up the specified width, and the remaining space will be split equally between the two margins:

```
.center {
  margin: auto;
  width: 50%;
  border: 3px solid green;
  padding: 10px;
}
```

Note: Center aligning has no effect if the `width` property is not set (or set to 100%).

Center Align Text

To just center the text inside an element, use `text-align: center;`

```
.center {
  text-align: center;
  border: 3px solid green;
}
```

Center an Image

To center an image, set left and right margin to `auto` and make it into a `block` element:

```
img {
  display: block;
  margin-left: auto;
  margin-right: auto;
  width: 40%;
}
```

Left and Right Align - Using position

One method for aligning elements is to use `position: absolute;`:

```
.right {
  position: absolute;
  right: 0px;
  width: 300px;
  border: 3px solid #73AD21;
  padding: 10px;
}
```

Left and Right Align - Using float

Another method for aligning elements is to use the `float` property:

```
.right {
  float: right;
  width: 300px;
  border: 3px solid #73AD21;
  padding: 10px;
}
```

Center Vertically - Using padding

There are many ways to center an element vertically in CSS. A simple solution is to use top and bottom `padding`:

```
.center {
  padding: 70px 0;
  border: 3px solid green;
}
```

To center both vertically and horizontally, use `padding` and `text-align: center`:

```
.center {
  padding: 70px 0;
  border: 3px solid green;
  text-align: center;
}
```

Center Vertically - Using line-height

Another trick is to use the `line-height` property with a value that is equal to the `height` property.

```
.center {
  line-height: 200px;
  height: 200px;
  border: 3px solid green;
  text-align: center;
}

/* If the text has multiple lines, add the following: */
.center p {
```

```
  line-height: 1.5;
  display: inline-block;
  vertical-align: middle;
}
```

Center Vertically - Using position & transform

If `padding` and `line-height` are not options, a third solution is to use positioning and the `transform` property:

```
.center {
  height: 200px;
  position: relative;
  border: 3px solid green;
}

.center p {
  margin: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

CSS Combinators

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 (~)

Descendant Selector

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

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

```
}
```

Child Selector

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

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

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 elements:

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

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 elements:

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

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 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 CSS Classes

Pseudo-classes can be combined with CSS classes:

When you hover over the link in the example, it will change color:

```
a.highlight:hover {
  color: #ff0000;
}
```

Hover on

An example of using the `:hover` pseudo-class on a element:

```
div:hover {
  background-color: blue;
}
```

Simple Tooltip Hover

Hover over a element to show a <p> element (like a tooltip):

```
p {
  display: none;
  background-color: yellow;
  padding: 20px;
}

div:hover p {
  display: block;
}
```

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:

```
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:

```
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:

```
p:first-child i {
  color: blue;
}
```

The :lang Pseudo-class

```
<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>
```

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;
}
```

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;
}
```

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:

```
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:

```
p.intro::first-letter {
```

```
  color: #ff0000;
  font-size:200%;
}
```

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:

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


p::first-line {
  color: #0000ff;
  font-variant: small-caps;
}
```

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);
```

```
}
```

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);
}
```

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;
}
```

All CSS Pseudo Elements

Selector	Example	Example description

::after	p::after	Insert something after the content of each <p> element
::before	p::before	Insert something before the content of each <p> element
::first-letter	p::first-letter	Selects the first letter of each <p> element
::first-line	p::first-line	Selects the first line of each <p> element
::selection	p::selection	Selects the portion of an element that is selected by a user

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

CSS Opacity

The `opacity` property specifies the opacity/transparency of an element.

The `opacity` property can take a value from 0.0 - 1.0. The lower value, the more transparent

```
img {
  opacity: 0.5;
  filter: alpha(opacity=50); /* For IE8 and earlier */
}
```

CSS Attribute Selectors

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

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;
}
```

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;
}
```

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;
}
```

CSS [attribute|="value"] Selector

The `[attribute|="value"]` selector is used to select elements with the specified attribute starting with the specified value.

The following example selects all elements with a class attribute value that begins with "top":

Note: The value has to be a whole word, either alone, like class="top", or followed by a hyphen(-), like class="top-text"!

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

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":

Note: The value does not have to be a whole word!

```
[class^="top"] {

  background: yellow;

}
```

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 ends with "test":

Note: The value does not have to be a whole word!

```
[class$="test"] {

  background: yellow;

}
```

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":

Note: The value does not have to be a whole word!

```
[class*="te"] {

  background: yellow;

}
```

CSS Specificity

What is Specificity?

If there are two or more conflicting CSS rules that point to the same element, the browser follows some rules to determine which one is most specific and therefore wins out.

Think of specificity as a score/rank that determines which style declarations are ultimately applied to an element.

The universal selector (*) has low specificity, while ID selectors are highly specific!

Specificity Hierarchy

Every selector has its place in the specificity hierarchy. There are four categories which define the specificity level of a selector:

Inline styles - An inline style is attached directly to the element to be styled. Example: <h1 style="color: #ffffff;">.

IDs - An ID is a unique identifier for the page elements, such as #navbar.

Classes, attributes and pseudo-classes - This category includes .classes, [attributes] and pseudo-classes such as :hover, :focus etc.

Elements and pseudo-elements - This category includes element names and pseudo-elements, such as h1, div, :before and :after.

ID selectors have a higher specificity than attribute selectors - Look at the following three code lines:

```
div#a {background-color: green;}
```

```
#a {background-color: yellow;}

div[id=a] {background-color: blue;}
```

Contextual selectors are more specific than a single element selector - The embedded style sheet is closer to the element to be styled. So in the following situation

From external CSS file:

```
#content h1 {background-color: red;}
```

In HTML file:

```
<style>

#content h1 {

  background-color: yellow;

}

</style>
```

A class selector beats any number of element selectors - a class selector such as .intro beats h1, p, div, etc:

```
.intro {background-color: yellow;}

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

CSS Gradients

CSS gradients let you display smooth transitions between two or more specified colors.

CSS defines two types of gradients:

- **Linear Gradients (goes down/up/left/right/diagonally)**
- **Radial Gradients (defined by their center)**

CSS Linear Gradients

To create a linear gradient you must define at least two color stops. Color stops are the colors you want to render smooth transitions among. You can also set a starting point and a direction (or an angle) along with the gradient effect.

Syntax

```
background-image: linear-gradient(direction, color-stop1, color-stop2, ...);
```

```
#grad {
  background-image: linear-gradient(red, yellow);
}
```

```
#grad {
  background-image: linear-gradient(to right, red , yellow);
}
```

```
#grad {
  background-image: linear-gradient(to bottom right, red, yellow);
}
```

Using Angles

If you want more control over the direction of the gradient, you can define an angle, instead of the predefined directions (to bottom, to top, to right, to left, to bottom right, etc.).

```
background-image: linear-gradient(angle, color-stop1, color-stop2);
```

```
#grad {
  background-image: linear-gradient(-90deg, red, yellow);
}
```

Repeating a linear-gradient

```
#grad {
  background-image: repeating-linear-gradient(red, yellow 10%, green 20%);
}
```

CSS Radial Gradients

```
background-image: radial-gradient(shape size at position, start-color, ..., last-color);
```

```
#grad {
  background-image: radial-gradient(red, yellow, green);
}
```

Radial Gradient - Differently Spaced Color Stops

```
#grad {
  background-image: radial-gradient(red 5%, yellow 15%, green 60%);
}
```

Set Shape

```
#grad {
  background-image: radial-gradient(circle, red, yellow, green);
}
```

Use of Different Size Keywords

The size parameter defines the size of the gradient. It can take four values:

- **closest-side**
- **farthest-side**
- **closest-corner**
- **farthest-corner**

```
#grad1 {

  background-image: radial-gradient(closest-side at 60% 55%, red,
yellow, black);

}



#grad2 {

  background-image: radial-gradient(farthest-side at 60% 55%, red,
yellow, black);

}
```

Repeating a radial-gradient

```
#grad {

  background-image: repeating-radial-gradient(red, yellow 10%, green
15%);

}
```

CSS Text Effects

In this chapter you will learn about the following properties:

- `text-overflow`
- `word-wrap`

- `word-break`
- `writing-mode`

CSS Text Overflow

The CSS `text-overflow` property specifies how overflowed content that is not displayed should be signaled to the user.

It can be clipped:

```
p.test1 {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: clip;
}


p.test2 {
      white-space: nowrap;
      width: 200px;
      border: 1px solid #000000;
      overflow: hidden;
      text-overflow: ellipsis;
}
```

CSS Word Wrapping

The CSS `word-wrap` property allows long words to be able to be broken and wrap onto the next line.

If a word is too long to fit within an area, it expands outside:

```
p {
  word-wrap: break-word;
}
```

CSS Word Breaking

The CSS `word-break` property specifies line breaking rules.

```
p.test1 {
  word-break: keep-all;
}


p.test2 {
  word-break: break-all;
}
```

CSS Writing Mode

The CSS `writing-mode` property specifies whether lines of text are laid out horizontally or vertically.

```
p.test1 {
  writing-mode: horizontal-tb;
}


span.test2 {
```

```
  writing-mode: vertical-rl;
}

p.test2 {
  writing-mode: vertical-rl;
}
```

CSS 2D Transforms

CSS Transforms

CSS transforms allow you to translate, rotate, scale, and skew elements.

A transformation is an effect that lets an element change shape, size and position.

CSS supports 2D and 3D transformations.

CSS 2D Transforms

In this chapter you will learn about the following 2D transformation methods:

- `translate()`
- `rotate()`
- `scale()`
- `skewX()`
- `skewY()`
- `matrix()`

The translate() Method

The `translate()` method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis).

The following example moves the element 50 pixels to the right, and 100 pixels down from its current position:

```
div {
  -ms-transform: translate(50px, 100px); /* IE 9 */
  -webkit-transform: translate(50px, 100px); /* Safari */
  transform: translate(50px, 100px);
}
```

The rotate() Method

The `rotate()` method rotates an element clockwise or counter-clockwise according to a given degree.

The following example rotates the element clockwise with 20 degrees:

```
div {
  -ms-transform: rotate(20deg); /* IE 9 */
  -webkit-transform: rotate(20deg); /* Safari */
  transform: rotate(20deg);
}
```

Using negative values will rotate the element counter-clockwise.

The following example rotates the element counter-clockwise with 20 degrees:

```
div {
  -ms-transform: rotate(-20deg); /* IE 9 */
  -webkit-transform: rotate(-20deg); /* Safari */
  transform: rotate(-20deg);
}
```

The scale() Method

The `scale()` method increases or decreases the size of an element (according to the parameters given for the width and height).

The following example increases the element to be two times of its original width, and three times of its original height:

```
div {
  -ms-transform: scale(2, 3); /* IE 9 */
  -webkit-transform: scale(2, 3); /* Safari */
  transform: scale(2, 3);
}
```

The following example decreases the element to be half of its original width and height:

```
div {
  -ms-transform: scale(0.5, 0.5); /* IE 9 */
  -webkit-transform: scale(0.5, 0.5); /* Safari */
  transform: scale(0.5, 0.5);
}
```

The skewX() Method

The `skewX()` method skews an element along the X-axis by the given angle.

The following example skews the element 20 degrees along the X-axis:

```
div {
  -ms-transform: skewX(20deg); /* IE 9 */
  -webkit-transform: skewX(20deg); /* Safari */
```

```
  transform: skewX(20deg);
}
```

The skewY() Method

The `skewY()` method skews an element along the Y-axis by the given angle.

The following example skews the element 20 degrees along the Y-axis:

```
div {
  -ms-transform: skewY(20deg); /* IE 9 */
  -webkit-transform: skewY(20deg); /* Safari */
  transform: skewY(20deg);
}
```

The skew() Method

The `skew()` method skews an element along the X and Y-axis by the given angles.

The following example skews the element 20 degrees along the X-axis, and 10 degrees along the Y-axis

```
div {
  -ms-transform: skew(20deg, 10deg); /* IE 9 */
  -webkit-transform: skew(20deg, 10deg); /* Safari */
  transform: skew(20deg, 10deg);
}
```

If the second parameter is not specified, it has a zero value. So, the following example skews the element 20 degrees along the X-axis:

```
div {
  -ms-transform: skew(20deg); /* IE 9 */
  -webkit-transform: skew(20deg); /* Safari */
  transform: skew(20deg);
}
```

The matrix() Method

The `matrix()` method combines all the 2D transform methods into one.

The matrix() method take six parameters, containing mathematic functions, which allows you to rotate, scale, move (translate), and skew elements.

The parameters are as follow:
matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY())

```
div {
  -ms-transform: matrix(1, -0.3, 0, 1, 0, 0); /* IE 9 */
  -webkit-transform: matrix(1, -0.3, 0, 1, 0, 0); /* Safari */
  transform: matrix(1, -0.3, 0, 1, 0, 0);
}
```

CSS 3D Transforms

CSS 3D Transforms

CSS allows you to format your elements using 3D transformations.

Mouse over the elements below to see the difference between a 2D and a 3D transformation:

In this chapter you will learn about the following 3D transformation methods:

- `rotateX()`
- `rotateY()`
- `rotateZ()`

The rotateX() Method

The `rotateX()` method rotates an element around its X-axis at a given degree:

```
#myDiv {

  -webkit-transform: rotateX(150deg); /* Safari */

  transform: rotateX(150deg);

}
```

The rotateY() Method

The `rotateY()` method rotates an element around its Y-axis at a given degree:

```
#myDiv {

  -webkit-transform: rotateY(130deg); /* Safari */

  transform: rotateY(130deg);

}
```

The rotateZ() Method

The `rotateZ()` method rotates an element around its Z-axis at a given degree:

```
#myDiv {
```

```
  -webkit-transform: rotateZ(90deg); /* Safari */

  transform: rotateZ(90deg);

}
```

CSS Transitions

CSS transitions allows you to change property values smoothly (from one value to another), over a given duration.

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 element. The element has also specified a transition effect for the width property, with a duration of 2 seconds:

```
div {

  width: 100px;

  height: 100px;

  background: red;

  -webkit-transition: width 2s; /* Safari */

  transition: width 2s;

}
```

he 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 element:

```
div:hover {

  width: 300px;

}
```

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 {

  -webkit-transition: width 2s, height 4s; /* Safari */

  transition: width 2s, height 4s;

}
```

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

```
#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;}
```

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:

```
div {

  -webkit-transition-delay: 1s; /* Safari */

  transition-delay: 1s;

}
```

CSS Animations

CSS animations allows animation of most HTML elements without using JavaScript or Flash!

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 element. The animation will last for 4 seconds, and it will gradually change the background-color of the element from "red" to "yellow":

```
/* The animation code */

@keyframes example {

  from {background-color: red;}

  to {background-color: yellow;}

}



/* The element to apply the animation to */

div {

  width: 100px;

  height: 100px;

  background-color: red;

  animation-name: example;

  animation-duration: 4s;

}
```

Note: The `animation-duration` property defines how long time 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 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;

}
```

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:

```
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:

```
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:

```
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
```

```
  animation-duration: 4s;

  animation-iteration-count: 3;

}
```

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):

```
div {

  width: 100px;

  height: 100px;

  position: relative;

  background-color: red;

  animation-name: example;

  animation-duration: 4s;
```

```
  animation-direction: 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 the some of the different speed curves that can be used:

```
#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;}
```

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 element retain the style values from the last keyframe when the animation ends:

```
div {

  width: 100px;

  height: 100px;

  background: red;

  position: relative;

  animation-name: example;

  animation-duration: 3s;
```

```
  animation-fill-mode: forwards;

}
```