CSS Links

With CSS, links can be styled in many different ways.

Text Link Text Link Link Button Link Button

Styling Links

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

Example

```
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

Example

```
/* 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:

Example

```
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:

Example

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

a:visited {
  background-color: cyan;
}

a:hover {
  background-color: lightgreen;
}

a:active {
  background-color: hotpink;
}
```

Link Buttons

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

Example

```
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;
}
```

More Examples

Example

This example demonstrates how to add other styles to hyperlinks:

```
a.one:link {color: #ff0000;}
a.one:visited {color: #0000ff;}
a.one:hover {color: #ffcc00;}

a.two:link {color: #ff0000;}
a.two:visited {color: #0000ff;}
a.two:hover {font-size: 150%;}

a.three:link {color: #ff0000;}
a.three:visited {color: #0000ff;}
a.three:hover {background: #66ff66;}

a.four:link {color: #ff0000;}
a.four:visited {color: #0000ff;}
a.four:hover {font-family: monospace;}

a.five:link {color: #ff0000; text-decoration: none;}
a.five:visited {color: #0000ff; text-decoration: none;}
a.five:hover {text-decoration: underline;}
```

Example

Another example of how to create link boxes/buttons:

```
a:link, a:visited {
  background-color: white;
  color: black;
  border: 2px solid green;
  padding: 10px 20px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
}

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

Example

This example demonstrates the different types of cursors (can be useful for links):

```
<span style="cursor: auto">auto</span><br>
<span style="cursor: crosshair">crosshair</span><br>
<span style="cursor: default">default</span><br>
```

```
<span style="cursor: e-resize">e-resize</span><br>
<span style="cursor: help">help</span><br>
<span style="cursor: move">move</span><br>
<span style="cursor: n-resize">n-resize</span><br>
<span style="cursor: ne-resize">ne-resize</span><br>
<span style="cursor: nw-resize">nw-resize</span><br>
<span style="cursor: pointer">pointer</span><br>
<span style="cursor: progress">progress</span><br>
<span style="cursor: s-resize">s-resize</span><br>
<span style="cursor: se-resize">se-resize</span><br>
<span style="cursor: sw-resize">sw-resize</span><br>
<span style="cursor: text">text</span><br>
<span style="cursor: w-resize">w-resize</span><br>
<span style="cursor: wait">wait</span>
```

CSS Lists

Unordered Lists:

- Coffee
- Tea
- Coca Cola

- Coffee
- Tea
- Coca Cola

Ordered Lists:

1. Coffee
2. Tea
3. Coca Cola

I. Coffee
II. Tea
III. Coca Cola

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:

Example

```
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;
}
```

Note: Some of the values are for unordered lists, and some for ordered lists.

An Image as The List Item Marker

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

Example

```
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:

- Coffee - A brewed drink prepared from roasted coffee beans...
- Tea
- Coca-cola

"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:

- Coffee - A brewed drink prepared from roasted coffee beans...
- Tea
- Coca-cola

Example

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

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>:

Example

```
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:

Example

```
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:

Example

```
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;
}
```

Result:

1. Coffee
2. Tea
3. Coca Cola

- Coffee
- Tea
- Coca Cola

More Examples

Customized list with a red left border
This example demonstrates how to create a list with a red left border.

Full-width bordered list
This example demonstrates how to create a bordered list without bullets.

All the different list-item markers for lists
This example demonstrates all the different list-item markers in CSS.

All CSS List Properties

Property	Description
list-style	Sets all the properties for a list in one declaration

list-style-image	Specifies an image as the list-item marker
list-style-position	Specifies the position of the list-item markers (bullet points)
list-style-type	Specifies the type of list-item marker

CSS Tables

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

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

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:

Example

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

Full-Width Table

The table above might seem small in some cases. If you need a table that should span the entire screen (full-width), add `width: 100%` to the <table> element:

Example

```
table {
  width: 100%;
}
```

Double Borders

Notice that the table in the examples above have double borders. This is because both the table and the <th> and <td> elements have separate borders.

To remove double borders, take a look at the example below.

Collapse Table Borders

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

Example

```
table {
  border-collapse: collapse;
}
```

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

Example

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

SS Table Size

Table Width and Height

The 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 70px:

Example

```
table {
  width: 100%;
}

th {
  height: 70px;
}
```

To create a table that should only span half the page, use `width: 50%`:

Example

```
table {
  width: 50%;
}

th {
  height: 70px;
}
```

CSS Table Alignment

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.

To center-align the content of <td> elements as well, use `text-align: center`:

Example

```
td {
  text-align: center;
}
```

To left-align the content, force the alignment of <th> elements to be left-aligned, with the `text-align: left` property:

Example

```
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:

Example

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

CSS Table Style

Table Padding

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

Example

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

Horizontal Dividers

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

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

Example

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

Hoverable Table

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

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

Example

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

Striped Tables

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

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

Example

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

Table Color

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

First Name	Last Name	Savings
Peter	Griffin	$100

Lois	Griffin	$150
Joe	Swanson	$300

Example

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

CSS Responsive Table

Responsive Table

A responsive table will display a horizontal scroll bar if the screen is too small to display the full content:

First Name	Last Name	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points
Jill	Smith	50	50	50	50	50	50	50	50	50	50	50	50
Eve	Jackson	94	94	94	94	94	94	94	94	94	94	94	94
Adam	Johnson	67	67	67	67	67	67	67	67	67	67	67	67

Add a container element (like) with `overflow-x:auto` around the <table> element to make it responsive:

Example

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

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

</div>
```

Note: In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).

More Examples

Make a fancy table
This example demonstrates how to create a fancy table.

Set the position of the table caption
This example demonstrates how to position the table caption.

CSS Table Properties

Property	Description
border	Sets all the border properties in one declaration
border-collapse	Specifies whether or not table borders should be collapsed
border-spacing	Specifies the distance between the borders of adjacent cells
caption-side	Specifies the placement of a table caption
empty-cells	Specifies whether or not to display borders and background on empty cells in a table
table-layout	Sets the layout algorithm to be used for a table

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`.

Click to show panel

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).

The element is a block-level element.

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:

Example

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

Note: Setting the display property of an element only changes **how the element is displayed**, NOT what kind of element it is. So, an inline element with `display: block;` is not allowed to have other block elements inside it.

The following example displays elements as block elements:

Example

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

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

Example

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

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

`display:none`

`visibility:hidden`

Reset

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:

Example

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

`visibility:hidden;` also hides an element.

However, the element will still take up the same space as before. The element will be hidden, but still affect the layout:

Example

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

CSS Display/Visibility Properties

Property	Description
display	Specifies how an element should be displayed
visibility	Specifies whether or not an element should be visible

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:

This element has a width of 500px, and margin set to auto.

Note: The problem with the `<div>` above occurs when the browser window is smaller than the width of the element. 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. This is important when making a site usable on small devices:

This element has a max-width of 500px, and margin set to auto.

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:

Example

```
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:

This element has position: static;

Here is the CSS that is used:

Example

```
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.

This element has position: relative;

Here is the CSS that is used:

Example

```
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:

Example

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

This element has `position: fixed;`

---

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`.

Here is a simple example:

This element has position: relative;
This element has position: absolute;

Here is the CSS that is used:

Example

```
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).

Note: Internet Explorer does not support sticky positioning. Safari requires a -webkit- prefix (see example below). You must also specify at least one of `top`, `right`, `bottom` or `left` for sticky positioning to work.

In this example, the sticky element sticks to the top of the page (`top: 0`), when you reach its scroll position.

Example

```
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:

This is a heading

Because the image has a z-index of -1, it will be placed behind the text.

Example

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

An element with greater stack order is always in front of an element with a lower stack order.

Note: If two positioned elements overlap without a `z-index` specified, the element positioned last in the HTML code will be shown on top.

Positioning Text In an Image

How to position text over an image:

Example

Bottom Left
Top Left
Top Right
Bottom Right
Centered

CSS Layout - Overflow

The CSS `overflow` property controls what happens to content that is too big to fit into an area.

This text is really long and the height of its container is only 100 pixels. Therefore, a scrollbar is added to help the reader to scroll the content. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem.

CSS Overflow

The `overflow` property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The `overflow` property has the following values:

- `visible` - Default. The overflow is not clipped. The content 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, and a scrollbar is added to see the rest of the content
- `auto` - Similar to `scroll`, but it adds scrollbars only when necessary

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:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```
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:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```
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):

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

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

---

overflow: auto

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

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```
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.

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

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

All CSS Overflow Properties

Property	Description
overflow	Specifies what happens if content overflows an element's box
overflow-x	Specifies what to do with the left/right edges of the content if it overflows the element's content area
overflow-y	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area

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.

Example - float: right;

The following example specifies that an image should float to the **right** in a text:

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

Example

```
img {
  float: right;
}
```

Example - float: left;

The following example specifies that an image should float to the **left** in a text:

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

Example

```
img {
  float: left;
}
```

Example - No float

In the following example the image will be displayed just where it occurs in the text (float: none;):

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

Example

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

Example - Float Next To Each Other

Normally div elements will be displayed on top of each other. However, if we use `float: left` we can let elements float next to each other:

Example

```
div {
  float: left;
  padding: 15px;
}

.div1 {
  background: red;
}
```

```
.div2 {
  background: yellow;
}

.div3 {
  background: green;
}
```

CSS Layout - clear and clearfix

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):

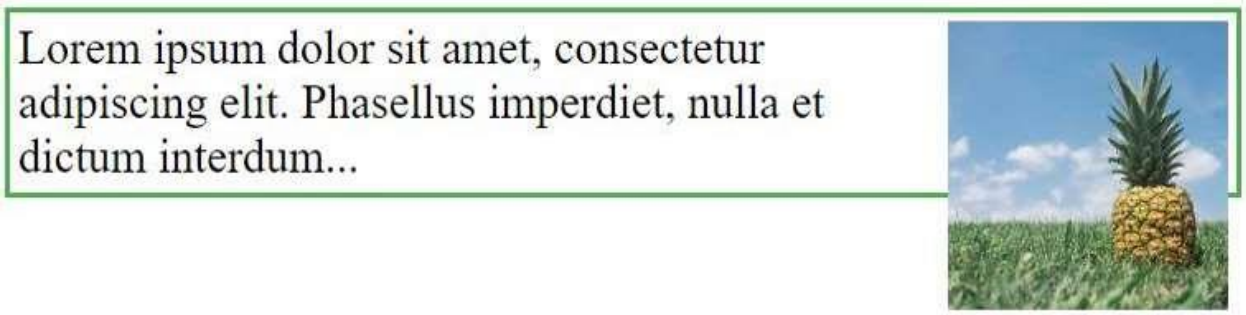
Example

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

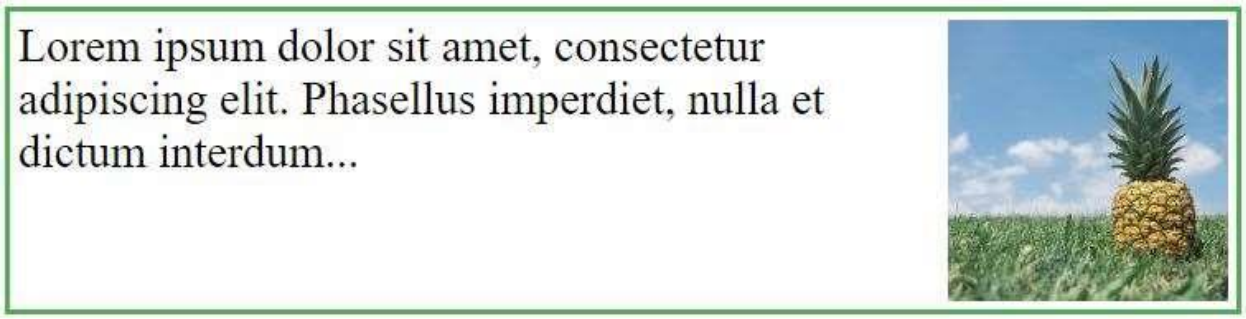
The clearfix Hack

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

Without Clearfix



With Clearfix



Then we can add `overflow: auto;` to the containing element to fix this problem:

Example

```
.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:

Example

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

CSS Layout - Float Examples

This page contains common float examples.

Grid of Boxes / Equal Width Boxes

Box 1

Box 2

Box 1

Box 2

Box 3

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

Example

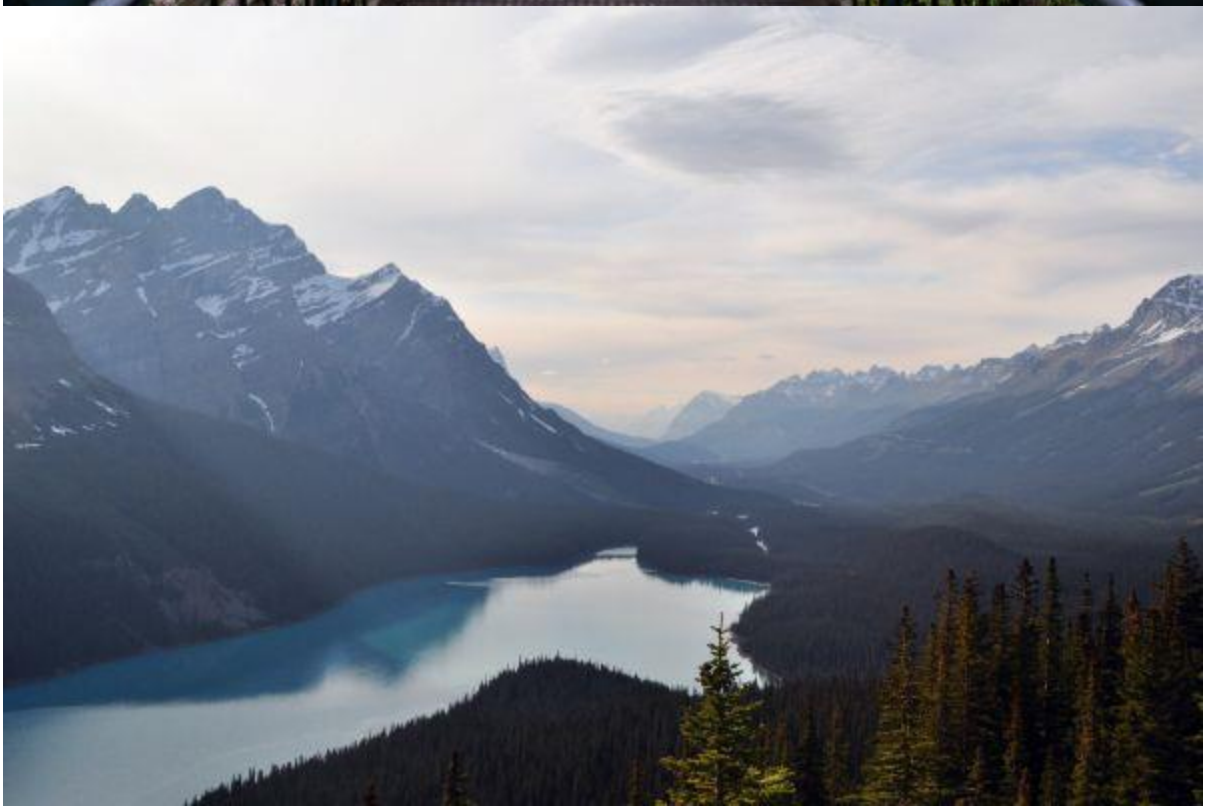
```
* {
  box-sizing: border-box;
}

.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 */
}
```

What is box-sizing?

You can easily create three floating boxes side by side. However, when you add something that enlarges the width of each box (e.g. padding or borders), the box will break. The `box-sizing` property allows us to include the padding and border in the box's total width (and height), making sure that the padding stays inside of the box and that it does not break.

You can read more about the box-sizing property in our CSS Box Sizing Chapter.

Images Side By Side



The grid of boxes can also be used to display images side by side:

Example

```
.img-container {
  float: left;
  width: 33.33%; /* three containers (use 25% for four, and 50% for two, etc) */
  padding: 5px; /* if you want space between the images */
}
```

Equal Height Boxes

In the previous example, you learned how to float boxes side by side with an equal width. However, it is not easy to create floating boxes with equal heights. A quick fix however, is to set a fixed height, like in the example below:

Box 1

Some content, some content, some content

Box 2

Some content, some content, some content

Some content, some content, some content

Some content, some content, some content

Example

```
.box {
  height: 500px;
}
```

However, this is not very flexible. It is ok if you can guarantee that the boxes will always have the same amount of content in them. But many times, the content is not the same. If you try the example above on a mobile phone, you will see that the second box's content will be displayed outside of the box. This is where CSS3 Flexbox comes in handy - as it can automatically stretch boxes to be as long as the longest box:

Example

Using **Flexbox** to create flexible boxes:

Box 1 - This is some text to make sure that the content gets really tall. This is some text to make sure that the content gets really tall. This is some text to make sure that the content gets really tall.
Box 2 - My height will follow Box 1.

The only problem with Flexbox is that it does not work in Internet Explorer 10 or earlier versions. You can read more about the Flexbox Layout Module in our CSS Flexbox Chapter.

Navigation Menu

Use `float` with a list of hyperlinks to create a horizontal menu:

Example

- Home
- News
- Contact
- About

Web Layout Example

It is also common to do entire web layouts using the `float` property:

Example

```
.header, .footer {
  background-color: grey;
  color: white;
  padding: 15px;
}

.column {
  float: left;
  padding: 15px;
}

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

```
.menu {
  width: 25%;
}

.content {
  width: 75%;
}
```

More Examples

An image with border and margins that floats to the right in a paragraph
Let an image float to the right in a paragraph. Add border and margins to the image.

An image with a caption that floats to the right
Let an image with a caption float to the right.

Let the first letter of a paragraph float to the left
Let the first letter of a paragraph float to the left and style the letter.

Creating a website with float
Use float to create a homepage with a navbar, header, footer, left content and main content.

All CSS Float Properties

Property	Description
box-sizing	Defines how the width and height of an element are calculated: should they include padding and borders, or not
clear	Specifies what elements can float beside the cleared element and on which side
float	Specifies how an element should float
overflow	Specifies what happens if content overflows an element's box
overflow-x	Specifies what to do with the left/right edges of the content if it overflows the element's content area
overflow-y	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area

CSS Layout - display: inline-block

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`:

Example

```
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;
}
```

Using inline-block to Create Navigation Links

One common use for `display: inline-block` is to display list items horizontally instead of vertically. The following example creates horizontal navigation links:

Example

```
.nav {
  background-color: yellow;
  list-style-type: none;
  text-align: center;
  padding: 0;
  margin: 0;
}

.nav li {
  display: inline-block;
  font-size: 20px;
  padding: 20px;
}
```

CSS Layout - Horizontal & Vertical Align

▲ ▼

◄ ►

Center elements horizontally and vertically

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:

This div element is centered.

Example

```
.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;`

This text is centered.

Example

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

Tip: For more examples on how to align text, see the CSS Text chapter.

Center an Image

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

Example

```
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;`:

In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since.

Example

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

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

Left and Right Align - Using float

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

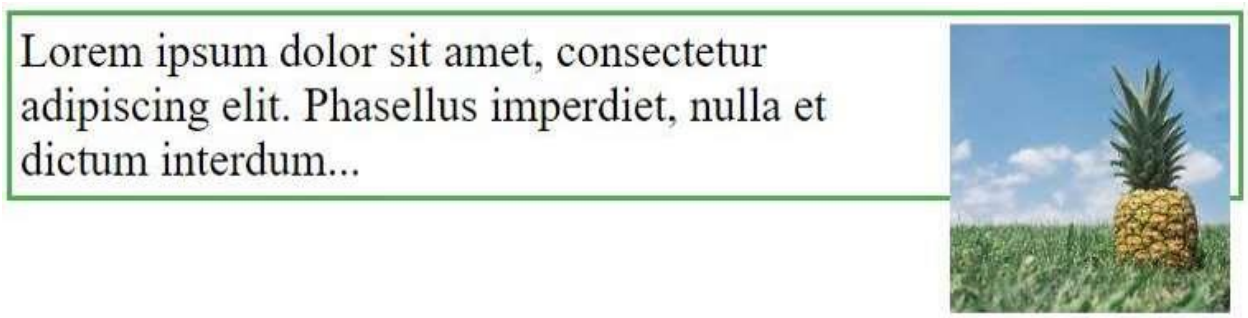
Example

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

The clearfix Hack

Note: If an element is taller than the element containing it, and it is floated, it will overflow outside of its container. You can use the "clearfix hack" to fix this (see example below).

Without Clearfix



With Clearfix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...

Then we can add the clearfix hack to the containing element to fix this problem:

Example

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

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`:

I am vertically centered.

Example

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

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

I am vertically and horizontally centered.

Example

```
.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:

I am vertically and horizontally centered.

Example

```
.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, another solution is to use positioning and the `transform` property:

I am vertically and horizontally centered.

Example

```
.center {
  height: 200px;
  position: relative;
```

```
  border: 3px solid green;
}

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

Tip: You will learn more about the transform property in our 2D Transforms Chapter.

Center Vertically - Using Flexbox

You can also use flexbox to center things. Just note that flexbox is not supported in IE10 and earlier versions:

I am vertically and horizontally centered.

Example

```
.center {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 200px;
  border: 3px solid green;
}
```

CSS Combinators

CSS Combinators

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 (~)

Descendant Selector

The descendant selector matches all elements that are descendants of a specified element.

The following example selects all <p> elements inside elements:

Example

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

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 element:

Example

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

Adjacent Sibling Selector (+)

The adjacent sibling selector is used to select an element that is directly after another specific element.

Sibling elements must have the same parent element, and "adjacent" means "immediately following".

The following example selects the first <p> element that are placed immediately after elements:

Example

```
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:

Example

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

All CSS Combinator Selectors

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

CSS Pseudo-classes

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

Mouse Over Me

Syntax

The syntax of pseudo-classes:

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

Anchor Pseudo-classes

Links can be displayed in different ways:

Example

```
/* 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:

Example

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

Hover on

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

Example

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

Simple Tooltip Hover

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

Hover over me to show the <p> element.

Example

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

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

CSS - The :first-child Pseudo-class

The `:first-child` pseudo-class matches a specified element that is the first child of another element.

Match the first <p> element

In the following example, the selector matches any <p> element that is the first child of any element:

Example

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

Match the first <i> element in all <p> elements

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

Example

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

Match all <i> elements in all first child <p> elements

In the following example, the selector matches all <i> elements in <p> elements that are the first child of another element:

Example

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

CSS - The :lang Pseudo-class

The `:lang` pseudo-class allows you to define special rules for different languages.

In the example below, `:lang` defines the quotation marks for <q> elements with lang="no":

Example

```
<html>
<head>
<style>
q:lang(no) {
  quotes: "~" "~";
}
</style>
</head>
<body>

<p>Some text <q lang="no">A quote in a paragraph</q> Some text.</p>

</body>
</html>
```

More Examples

Add different styles to hyperlinks
This example demonstrates how to add other styles to hyperlinks.

Use of :focus
This example demonstrates how to use the :focus pseudo-class.

All CSS Pseudo Classes

Selector	Example	Example description

:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a specified range
:invalid	input:invalid	Selects all <input> elements with an invalid value
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"
:last-child	p:last-child	Selects every <p> elements that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> element
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child

:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a specified range
:read-only	input:read-only	Selects <input> elements with a "readonly" attribute specified
:read-write	input:read-write	Selects <input> elements with no "readonly" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified
:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

All CSS Pseudo Elements

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

CSS Pseudo-elements

What are Pseudo-Elements?

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

Syntax

The syntax of pseudo-elements:

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

The ::first-line Pseudo-element

The `::first-line` pseudo-element is used to add a special style to the first line of a text.

The following example formats the first line of the text in all <p> elements:

Example

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

Note: The `::first-line` pseudo-element can only be applied to block-level elements.

The following properties apply to the `::first-line` pseudo-element:

- font properties
- color properties

- background properties
- word-spacing
- letter-spacing
- text-decoration
- vertical-align
- text-transform
- line-height
- clear

Notice the double colon notation - `::first-line` versus `:first-line`

The double colon replaced the single-colon notation for pseudo-elements in CSS3. This was an attempt from W3C to distinguish between **pseudo-classes** and **pseudo-elements**.

The single-colon syntax was used for both pseudo-classes and pseudo-elements in CSS2 and CSS1.

For backward compatibility, the single-colon syntax is acceptable for CSS2 and CSS1 pseudo-elements.

The ::first-letter Pseudo-element

The `::first-letter` pseudo-element is used to add a special style to the first letter of a text.

The following example formats the first letter of the text in all <p> elements:

Example

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

Note: The `::first-letter` pseudo-element can only be applied to block-level elements.

The following properties apply to the ::first-letter pseudo- element:

- font properties
- color properties
- background properties
- margin properties
- padding properties
- border properties

- text-decoration
- vertical-align (only if "float" is "none")
- text-transform
- line-height
- float
- clear

Pseudo-elements and CSS Classes

Pseudo-elements can be combined with CSS classes:

Example

```
p.intro::first-letter {
  color: #ff0000;
  font-size: 200%;
}
```

The example above will display the first letter of paragraphs with class="intro", in red and in a larger size.

Multiple Pseudo-elements

Several pseudo-elements can also be combined.

In the following example, the first letter of a paragraph will be red, in an xx-large font size. The rest of the first line will be blue, and in small-caps. The rest of the paragraph will be the default font size and color:

Example

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

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

CSS - The ::before Pseudo-element

The `::before` pseudo-element can be used to insert some content before the content of an element.

The following example inserts an image before the content of each <h1> element:

Example

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

CSS - The ::after Pseudo-element

The `::after` pseudo-element can be used to insert some content after the content of an element.

The following example inserts an image after the content of each <h1> element:

Example

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

CSS - The ::marker Pseudo-element

The `::marker` pseudo-element selects the markers of list items.

The following example styles the markers of list items:

Example

```
::marker {
  color: red;
  font-size: 23px;
}
```

CSS - The ::selection Pseudo-element

The `::selection` pseudo-element matches the portion of an element that is selected by a user.

The following CSS properties can be applied to `::selection`: `color`, `background`, `cursor`, and `outline`.

The following example makes the selected text red on a yellow background:

Example

```
::selection {
  color: red;
  background: yellow;
}
```

CSS Opacity / Transparency

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

Transparent Image

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

opacity 0.2

opacity 0.5

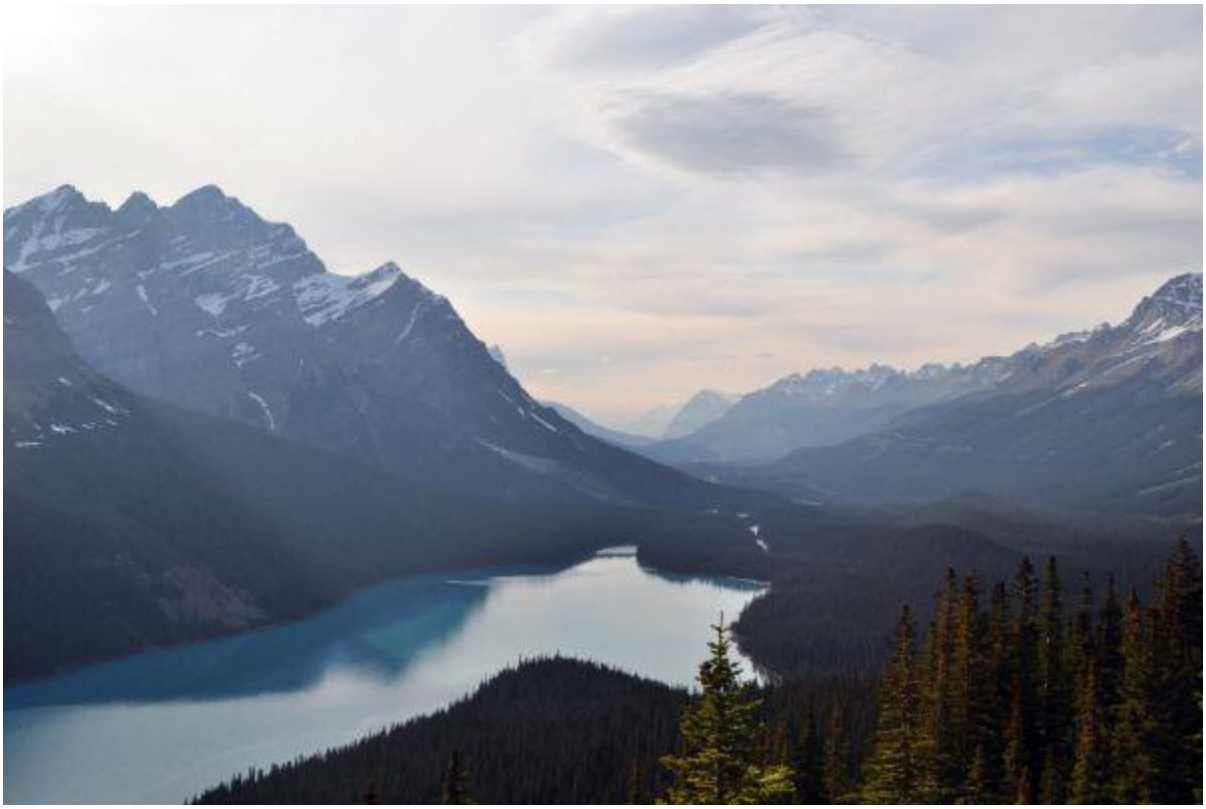
opacity 1
(default)

Example

```
img {
  opacity: 0.5;
}
```

Transparent Hover Effect

The `opacity` property is often used together with the `:hover` selector to change the opacity on mouse-over:



Example

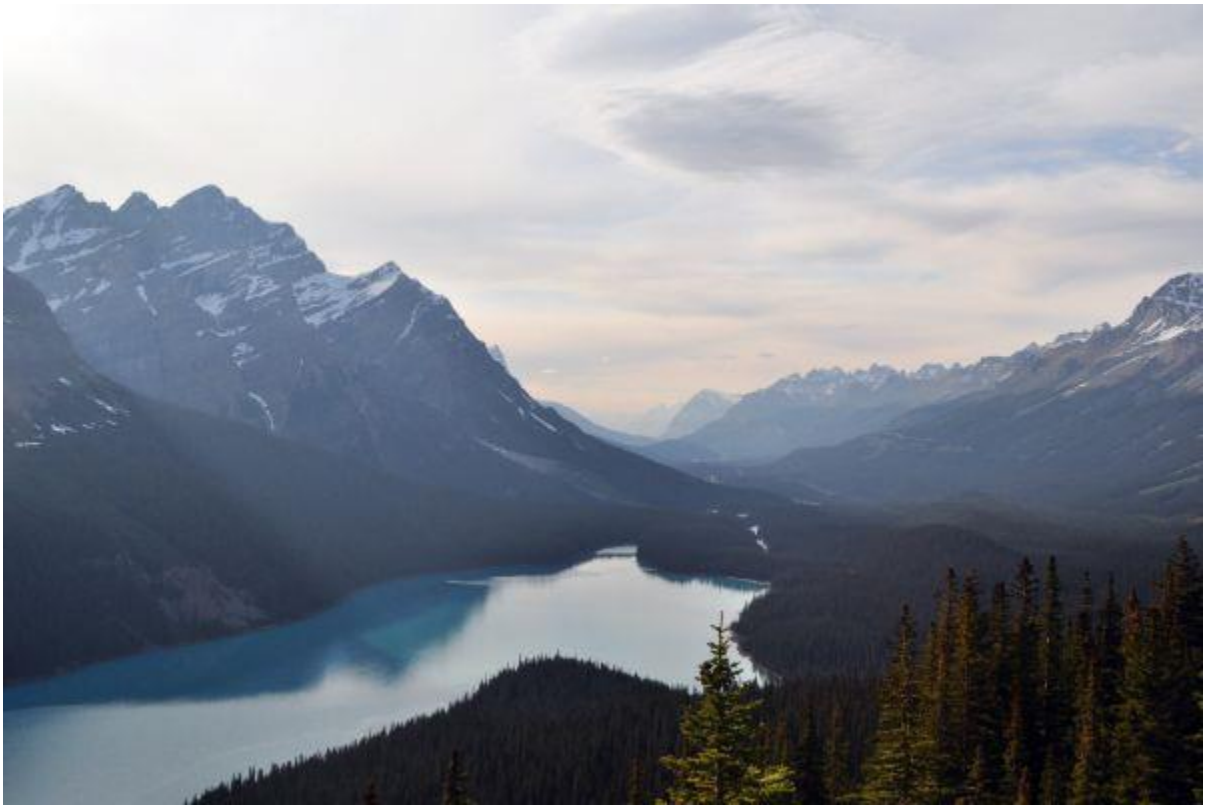
```
img {
  opacity: 0.5;
}

img:hover {
  opacity: 1.0;
}
```

Example explained

The first CSS block is similar to the code in Example 1. In addition, we have added what should happen when a user hovers over one of the images. In this case we want the image to NOT be transparent when the user hovers over it. The CSS for this is `opacity:1;`.

When the mouse pointer moves away from the image, the image will be transparent again.

An example of reversed hover effect:



Example

```
img:hover {
  opacity: 0.5;
}
```

Transparent Box

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

opacity 1

opacity 0.6

opacity 0.3

opacity 0.1

Example

```
div {
  opacity: 0.3;
}
```

Transparency using RGBA

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

100% opacity

60% opacity

30% opacity

10% opacity

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

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

Tip: You will learn more about RGBA Colors in our CSS Colors Chapter.

Example

```
div {
  background: rgba(76, 175, 80, 0.3) /* Green background with 30% opacity */
}
```

Text in Transparent Box

This is some text that is placed in the transparent box.

Example

```
<html>
<head>
<style>
div.background {
  background: url(klematis.jpg) repeat;
  border: 2px solid black;
}

div.transbox {
  margin: 30px;
  background-color: #ffffff;
  border: 1px solid black;
  opacity: 0.6;
}

div.transbox p {
  margin: 5%;
  font-weight: bold;
  color: #000000;
}
</style>
</head>
<body>

<div class="background">
  <div class="transbox">
    <p>This is some text that is placed in the transparent box.</p>
  </div>
</div>

</body>
</html>
```

Example explained

First, we create a element (class="background") with a background image, and a border.

Then we create another (class="transbox") inside the first .

The have a background color, and a border - the div is transparent.

Inside the transparent , we add some text inside a <p> element.

CSS Navigation Bar

Demo: Navigation Bars

Vertical

- Home
- News
- Contact
- About

Horizontal

- Home
- News
- Contact
- About

- Home
- News
- Contact
- About

Navigation Bars

Having easy-to-use navigation is important for any web site.

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

Navigation Bar = List of Links

A navigation bar needs standard HTML as a base.

In our examples we will build the navigation bar from a standard HTML list.

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

Example

```
<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>
```

Now let's remove the bullets and the margins and padding from the list:

Example

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

Example explained:

- `list-style-type: none;` - Removes the bullets. A navigation bar does not need list markers
- Set `margin: 0;` and `padding: 0;` to remove browser default settings

The code in the example above is the standard code used in both vertical, and horizontal navigation bars, which you will learn more about in the next chapters.

CSS Vertical Navigation Bar

Vertical Navigation Bar

- Home
- News
- Contact
- About

To build a vertical navigation bar, you can style the <a> elements inside the list, in addition to the code from the previous page:

Example

```
li a {
  display: block;
  width: 60px;
}
```

Example explained:

- `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

You can also set the width of <ul>, and remove the width of <a>, as they will take up the full width available when displayed as block elements. This will produce the same result as our previous example:

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 60px;
}

li a {
  display: block;
}
```

Vertical Navigation Bar Examples

Create a basic vertical navigation bar with a gray background color and change the background color of the links when the user moves the mouse over them:

- Home
- News
- Contact
- About

Example

```css
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 200px;
  background-color: #f1f1f1;
}

li a {
  display: block;
  color: #000;
  padding: 8px 16px;
  text-decoration: none;
}

/* Change the link color on hover */
li a:hover {
  background-color: #555;
  color: white;
}
```

Active/Current Navigation Link

Add an "active" class to the current link to let the user know which page he/she is on:

- Home
- News
- Contact
- About

Example

```css
.active {
  background-color: #4CAF50;
  color: white;
}
```

Center Links & Add Borders

Add `text-align:center` to <li> or <a> to center the links.

Add the `border` property to <ul> add a border around the navbar. If you also want borders inside the navbar, add a `border-bottom` to all <li> elements, except for the last one:

- Home
- News
- Contact
- About

Example

```
ul {
  border: 1px solid #555;
}

li {
  text-align: center;
  border-bottom: 1px solid #555;
}

li:last-child {
  border-bottom: none;
}
```

Full-height Fixed Vertical Navbar

Create a full-height, "sticky" side navigation:

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 25%;
  background-color: #f1f1f1;
  height: 100%; /* Full height */
  position: fixed; /* Make it stick, even on scroll */
  overflow: auto; /* Enable scrolling if the sidenav has too much content */
}
```

Note: This example might not work properly on mobile devices. CSS

Horizontal Navigation Bar

Horizontal Navigation Bar

- Home
- News
- Contact
- About

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

Inline List Items

One way to build a horizontal navigation bar is to specify the <li> elements as inline, in addition to the "standard" code from the previous page:

Example

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

Example explained:

- `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

Floating List Items

Another way of creating a horizontal navigation bar is to float the <li> elements, and specify a layout for the navigation links:

Example

```
li {
  float: left;
}

a {
  display: block;
  padding: 8px;
  background-color: #dddddd;
}
```

Example explained:

- `float: left;` - use float to get block elements to slide next to each other
- `display: block;` - Allows us to specify padding (and height, width, margins, etc. if you want)

- `padding: 8px;` - Since block elements take up the full width available, they cannot float next to each other. Therefore, specify some padding to make them look good
- `background-color: #dddddd;` - Add a gray background-color to each a element

Tip: Add the background-color to <ul> instead of each <a> element if you want a full-width background color:

Example

```
ul {
  background-color: #dddddd;
}
```

Horizontal Navigation Bar Examples

Create a basic horizontal navigation bar with a dark background color and change the background color of the links when the user moves the mouse over them:

- Home
- News
- Contact
- About

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

li {
  float: left;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
```

```
/* Change the link color to #111 (black) on hover */
li a:hover {
  background-color: #111;
}
```

Active/Current Navigation Link

Add an "active" class to the current link to let the user know which page he/she is on:

- Home
- News
- Contact
- About

Example

```
.active {
  background-color: #4CAF50;
}
```

Right-Align Links

Right-align links by floating the list items to the right (`float:right;`):

- Home
- News
- Contact
- About

Example

```
<ul>
  <li><a href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li><a href="#contact">Contact</a></li>
  <li style="float:right"><a class="active" href="#about">About</a></li>
</ul>
```

Border Dividers

Add the `border-right` property to <li> to create link dividers:

- Home

- News
- Contact
- About

Example

```
/* Add a gray right border to all list items, except the last item (last-child) */
li {
  border-right: 1px solid #bbb;
}

li:last-child {
  border-right: none;
}
```

Fixed Navigation Bar

Make the navigation bar stay at the top or the bottom of the page, even when the user scrolls the page:

Fixed Top

```
ul {
  position: fixed;
  top: 0;
  width: 100%;
}
```

Fixed Bottom

```
ul {
  position: fixed;
  bottom: 0;
  width: 100%;
}
```

Note: Fixed position might not work properly on mobile devices.

Gray Horizontal Navbar

An example of a gray horizontal navigation bar with a thin gray border:

- Home
- News

- Contact
- About

Example

```
ul {
  border: 1px solid #e7e7e7;
  background-color: #f3f3f3;
}

li a {
  color: #666;
}
```

Sticky Navbar

Add `position: sticky;` to <ul> to create a sticky navbar.

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).

Example

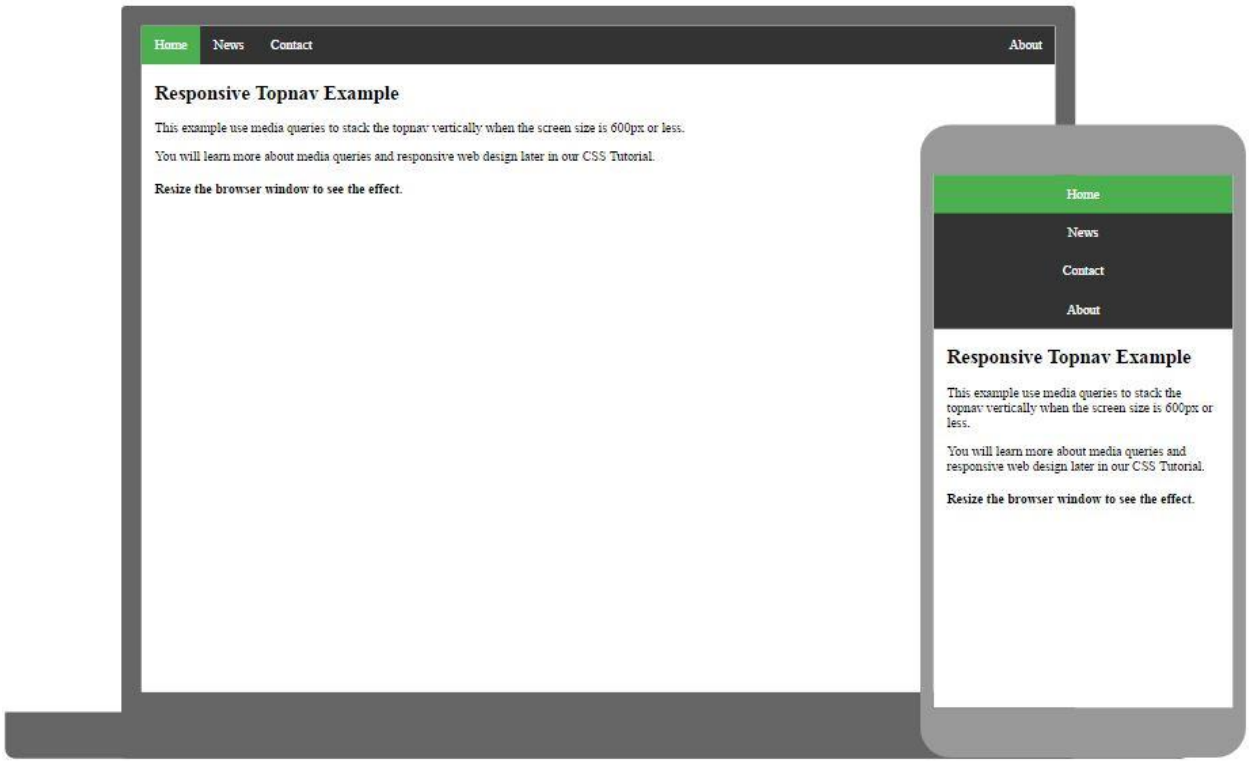
```
ul {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
}
```

Note: Internet Explorer do not support sticky positioning. Safari requires a -webkit- prefix (see example above). You must also specify at least one of `top`, `right`, `bottom` or `left` for sticky positioning to work.

More Examples

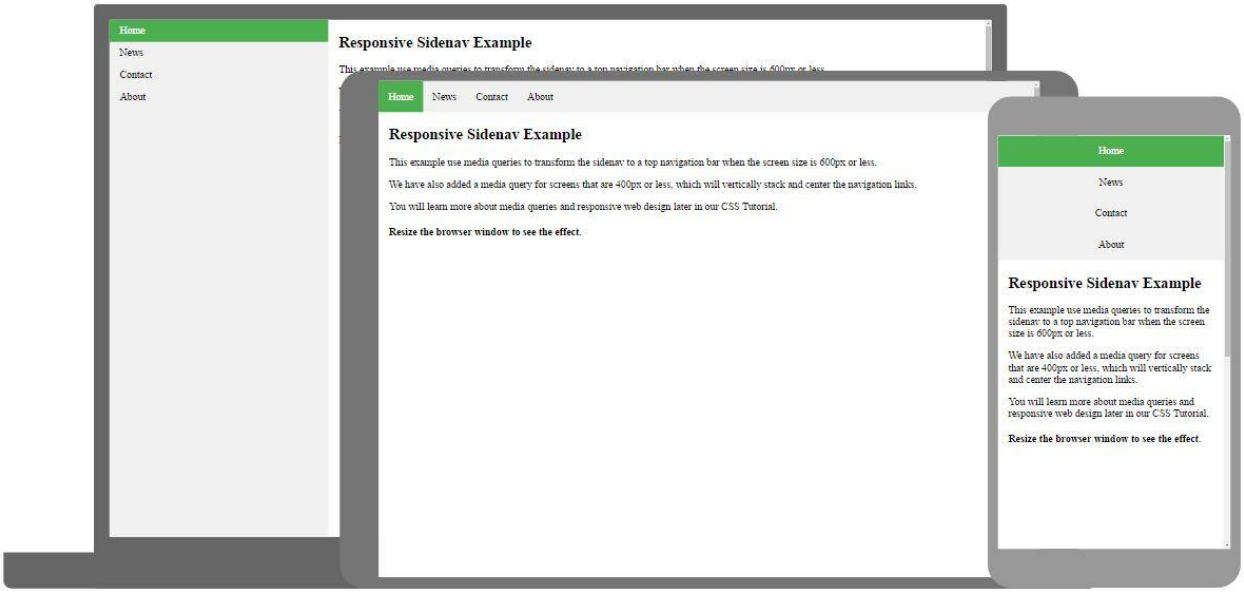
Responsive Topnav

How to use CSS media queries to create a responsive top navigation.



Responsive Sidenav

How to use CSS media queries to create a responsive side navigation.



Dropdown Navbar

How to add a dropdown menu inside a navigation bar.

CSS Dropdowns

Create a hoverable dropdown with CSS.

Demo: Dropdown Examples

Move the mouse over the examples below:

Dropdown Text

Other: 

Basic Dropdown

Create a dropdown box that appears when the user moves the mouse over an element.

Example

```
<style>
.dropdown {
  position: relative;
  display: inline-block;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  padding: 12px 16px;
  z-index: 1;
}
```

```
.dropdown:hover .dropdown-content {
  display: block;
}
</style>

<div class="dropdown">
  <span>Mouse over me</span>
  <div class="dropdown-content">
    <p>Hello World!</p>
  </div>
</div>
```

Example Explained

HTML) Use any element to open the dropdown content, e.g. a , or a <button> element.

Use a container element (like) to create the dropdown content and add whatever you want inside of it.

Wrap a element around the elements to position the dropdown content correctly with CSS.

CSS) The `.dropdown` class uses `position:relative`, which is needed when we want the dropdown content to be placed right below the dropdown button (using `position:absolute`).

The `.dropdown-content` class holds the actual dropdown content. It is hidden by default, and will be displayed on hover (see below). Note the `min-width` is set to 160px. Feel free to change this. **Tip:** If you want the width of the dropdown content to be as wide as the dropdown button, set the `width` to 100% (and `overflow:auto` to enable scroll on small screens).

Instead of using a border, we have used the CSS `box-shadow` property to make the dropdown menu look like a "card".

The `:hover` selector is used to show the dropdown menu when the user moves the mouse over the dropdown button.

Dropdown Menu

Create a dropdown menu that allows the user to choose an option from a list:

This example is similar to the previous one, except that we add links inside the dropdown box and style them to fit a styled dropdown button:

Example

```
<style>
/* Style The Dropdown Button */
.dropbtn {
  background-color: #4CAF50;
  color: white;
  padding: 16px;
  font-size: 16px;
  border: none;
  cursor: pointer;
}

/* The container <div> - needed to position the dropdown content */
.dropdown {
  position: relative;
  display: inline-block;
}

/* Dropdown Content (Hidden by Default) */
.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}

/* Links inside the dropdown */
.dropdown-content a {
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
}

/* Change color of dropdown links on hover */
.dropdown-content a:hover {background-color: #f1f1f1}
```

```
/* Show the dropdown menu on hover */
.dropdown:hover .dropdown-content {
  display: block;
}

/* Change the background color of the dropdown button when the dropdown content is shown */
.dropdown:hover .dropbtn {
  background-color: #3e8e41;
}
</style>

<div class="dropdown">
  <button class="dropbtn">Dropdown</button>
  <div class="dropdown-content">
    <a href="#">Link 1</a>
    <a href="#">Link 2</a>
    <a href="#">Link 3</a>
  </div>
</div>
```

Right-aligned Dropdown Content

If you want the dropdown menu to go from right to left, instead of left to right, add `right: 0;`

Example

```
.dropdown-content {
  right: 0;
}
```

More Examples

Dropdown Image

How to add an image and other content inside the dropdown box.

Hover over the image:

Dropdown Navbar

How to add a dropdown menu inside a navigation bar.

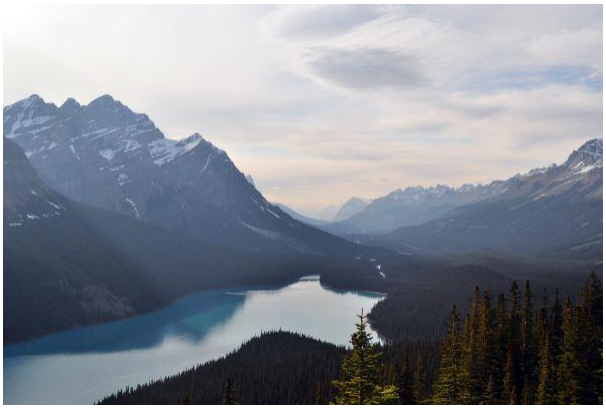
CSS Image Gallery

CSS can be used to create an image gallery.

Add a description of the image here

Add a description of the image here

Add a description of the image here



Add a description of the image here

Image Gallery

The following image gallery is created with CSS:

Example

```
<html>
<head>
<style>
div.gallery {
  margin: 5px;
  border: 1px solid #ccc;
  float: left;
  width: 180px;
}
```

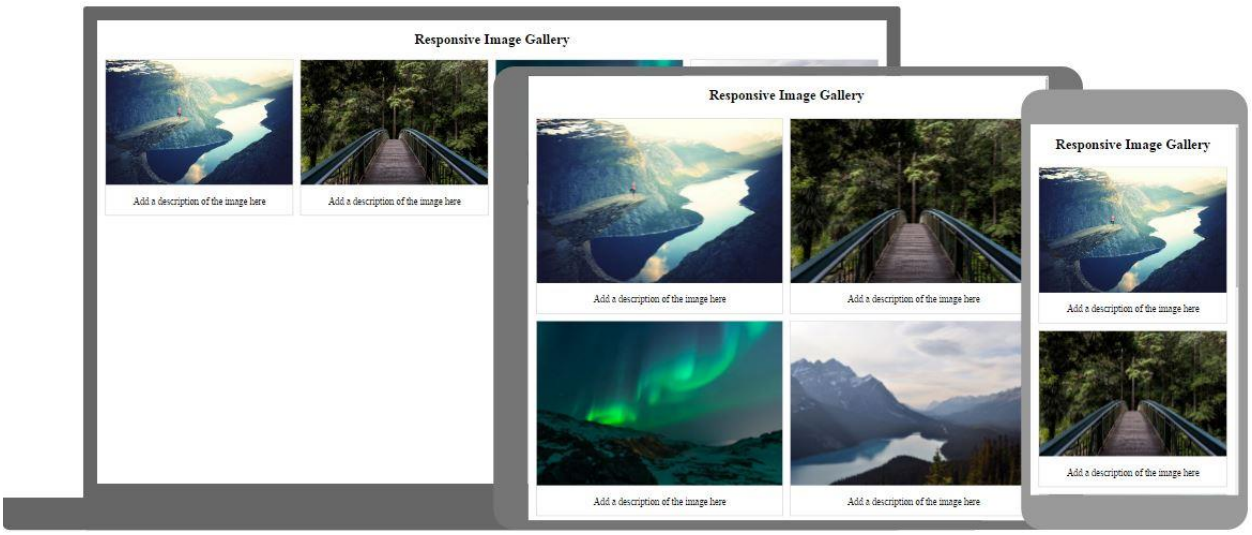
```
div.gallery:hover {
  border: 1px solid #777;
}

div.gallery img {
  width: 100%;
  height: auto;
}

div.desc {
  padding: 15px;
  text-align: center;
}
</style>
</head>
<body>

<div class="gallery">
  <a target="_blank" href="img_5terre.jpg">
    <img src="img_5terre.jpg" alt="Cinque Terre" width="600" height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_forest.jpg">
    <img src="img_forest.jpg" alt="Forest" width="600" height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_lights.jpg">
    <img src="img_lights.jpg" alt="Northern Lights" width="600" height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_mountains.jpg">
    <img src="img_mountains.jpg" alt="Mountains" width="600" height="400">
```

```
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

</body>
</html>
```

More Examples

Responsive Image Gallery

How to use CSS media queries to create a responsive image gallery that will look good on desktops, tablets and smart phones.



CSS Image Sprites

Image Sprites

An image sprite is a collection of images put into a single image.

A web page with many images can take a long time to load and generates multiple server requests.

Using image sprites will reduce the number of server requests and save bandwidth.

Image Sprites - Simple Example

Instead of using three separate images, we use this single image ("img_navsprites.gif"):

With CSS, we can show just the part of the image we need.

In the following example the CSS specifies which part of the "img_navsprites.gif" image to show:

Example

```
#home {
  width: 46px;
  height: 44px;
  background: url(img_navsprites.gif) 0 0;
}
```

Example explained:

- `<img id="home" src="img_trans.gif">` - Only defines a small transparent image because the src attribute cannot be empty. The displayed image will be the background image we specify in CSS
- `width: 46px; height: 44px;` - Defines the portion of the image we want to use
- `background: url(img_navsprites.gif) 0 0;` - Defines the background image and its position (left 0px, top 0px)

This is the easiest way to use image sprites, now we want to expand it by using links and hover effects.

Image Sprites - Create a Navigation List

We want to use the sprite image ("img_navsprites.gif") to create a navigation list.

We will use an HTML list, because it can be a link and also supports a background image:

Example

```
#navlist {
  position: relative;
}

#navlist li {
  margin: 0;
  padding: 0;
  list-style: none;
  position: absolute;
  top: 0;
}

#navlist li, #navlist a {
  height: 44px;
  display: block;
}

#home {
  left: 0px;
  width: 46px;
  background: url('img_navsprites.gif') 0 0;
}

#prev {
  left: 63px;
  width: 43px;
  background: url('img_navsprites.gif') -47px 0;
}

#next {
  left: 129px;
  width: 43px;
  background: url('img_navsprites.gif') -91px 0;
}
```

Example explained:

- #navlist {position:relative;} - position is set to relative to allow absolute positioning inside it
- #navlist li {margin:0;padding:0;list-style:none;position:absolute;top:0;} - margin and padding are set to 0, list-style is removed, and all list items are absolute positioned
- #navlist li, #navlist a {height:44px;display:block;} - the height of all the images are 44px

Now start to position and style for each specific part:

- #home {left:0px;width:46px;} - Positioned all the way to the left, and the width of the image is 46px
- #home {background:url(img_navsprites.gif) 0 0;} - Defines the background image and its position (left 0px, top 0px)
- #prev {left:63px;width:43px;} - Positioned 63px to the right (#home width 46px + some extra space between items), and the width is 43px.
- #prev {background:url('img_navsprites.gif') -47px 0;} - Defines the background image 47px to the right (#home width 46px + 1px line divider)
- #next {left:129px;width:43px;}- Positioned 129px to the right (start of #prev is 63px + #prev width 43px + extra space), and the width is 43px.
- #next {background:url('img_navsprites.gif') -91px 0;} - Defines the background image 91px to the right (#home width 46px + 1px line divider + #prev width 43px + 1px line divider)

Image Sprites - Hover Effect

Now we want to add a hover effect to our navigation list.

Tip: The `:hover` selector can be used on all elements, not only on links.

Our new image ("img_navsprites_hover.gif") contains three navigation images and three images to use for hover effects:

Because this is one single image, and not six separate files, there will be **no loading delay** when a user hovers over the image.

We only add three lines of code to add the hover effect:

Example

```
#home a:hover {
  background: url('img_navsprites_hover.gif') 0 -45px;
}

#prev a:hover {
  background: url('img_navsprites_hover.gif') -47px -45px;
}
```

```
#next a:hover {
  background: url('img_navsprites_hover.gif') -91px -45px;
}
```

CSS Attribute Selectors

Style HTML Elements With Specific Attributes

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

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:

Example

```
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:

Example

```
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":

Example

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

The example above will match elements with title="flower", title="summer flower", and title="flower new", but not title="my-flower" or title="flowers".

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"!

Example

```
[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!

Example

```
[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!

Example

```
[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!

Example

```
[class*="te"] {
  background: yellow;
}
```

Styling Forms

The attribute selectors can be useful for styling forms without class or ID:

Example

```
input[type="text"] {
  width: 150px;
  display: block;
  margin-bottom: 10px;
  background-color: yellow;
}

input[type="button"] {
  width: 120px;
  margin-left: 35px;
  display: block;
}
```

All 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"

CSS Forms

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

First Name Last Name Country
Try it Yourself »

Styling Input Fields

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

First Name

Example

```
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
- etc..

Padded Inputs

Use the `padding` property to add space inside the text field.

Tip: When you have many inputs after each other, you might also want to add some `margin`, to add more space outside of them:

First Name Last Name

Example

```
input[type=text] {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  box-sizing: border-box;
}
```

Note that we have set the `box-sizing` property to `border-box`. This makes sure that the padding and eventually borders are included in the total width and height of the elements.
Read more about the `box-sizing` property in our CSS Box Sizing chapter.

Bordered Inputs

Use the `border` property to change the border size and color, and use the `border-radius` property to add rounded corners:

First Name

Example

```
input[type=text] {
  border: 2px solid red;
  border-radius: 4px;
}
```

If you only want a bottom border, use the `border-bottom` property:

First Name

Example

```
input[type=text] {
  border: none;
  border-bottom: 2px solid red;
}
```

Colored Inputs

Use the `background-color` property to add a background color to the input, and the `color` property to change the text color:

John

Example

```
input[type=text] {
  background-color: #3CBC8D;
  color: white;
}
```

Focused Inputs

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:

Example

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

Example

```
input[type=text]:focus {
  border: 3px solid #555;
}
```

Input with icon/image

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:

Example

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

Animated Search Input

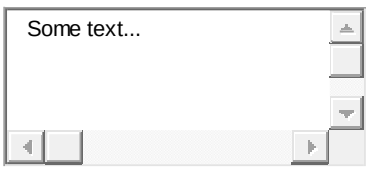
In this example we use the CSS `transition` property to animate the width of the search input when it gets focus. You will learn more about the `transition` property later, in our CSS Transitions chapter.

Example

```
input[type=text] {
  transition: width 0.4s ease-in-out;
}

input[type=text]:focus {
  width: 100%;
}
```

Styling Textareas

Tip: Use the `resize` property to prevent textareas from being resized (disable the "grabber" in the bottom right corner):



Example

```
textarea {
  width: 100%;
  height: 150px;
  padding: 12px 20px;
  box-sizing: border-box;
  border: 2px solid #ccc;
  border-radius: 4px;
  background-color: #f8f8f8;
  resize: none;
}
```

Styling Select Menus

Example

```
select {
  width: 100%;
  padding: 16px 20px;
  border: none;
  border-radius: 4px;
  background-color: #f1f1f1;
}
```

Styling Input Buttons

Example

```
input[type=button], input[type=submit], input[type=reset] {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 16px 32px;
  text-decoration: none;
  margin: 4px 2px;
  cursor: pointer;
}

/* Tip: use width: 100% for full-width buttons */
```

For more information about how to style buttons with CSS, read our CSS Buttons Tutorial.

Responsive Form

Resize the browser window to see the effect. When the screen is less than 600px wide, make the two columns stack on top of each other instead of next to each other.

CSS Counters

Pizza

Hamburger

Hotdogs

CSS counters are "variables" maintained by CSS whose values can be incremented by CSS rules (to track how many times they are used). Counters let you adjust the appearance of content based on its placement in the document.

Automatic Numbering With Counters

CSS counters are like "variables". The variable values can be incremented by CSS rules (which will track how many times they are used).

To work with CSS counters we will use the following properties:

- `counter-reset` - Creates or resets a counter
- `counter-increment` - Increments a counter value
- `content` - Inserts generated content
- `counter()` or `counters()` function - Adds the value of a counter to an element

To use a CSS counter, it must first be created with `counter-reset`.

The following example creates a counter for the page (in the body selector), then increments the counter value for each <h2> element and adds "Section <*value of the counter*>:" to the beginning of each <h2> element:

Example

```
body {
  counter-reset: section;
}
```

```
h2::before {
  counter-increment: section;
  content: "Section " counter(section) ": ";
}
```

Nesting Counters

The following example creates one counter for the page (section) and one counter for each <h1> element (subsection). The "section" counter will be counted for each <h1> element with "Section <*value of the section counter*>.", and the "subsection" counter will be counted for each <h2> element with "<*value of the section counter*>.<*value of the subsection counter*>":

Example

```
body {
  counter-reset: section;
}

h1 {
  counter-reset: subsection;
}

h1::before {
  counter-increment: section;
  content: "Section " counter(section) ". ";
}

h2::before {
  counter-increment: subsection;
  content: counter(section) "." counter(subsection) " ";
}
```

A counter can also be useful to make outlined lists because a new instance of a counter is automatically created in child elements. Here we use the `counters()` function to insert a string between different levels of nested counters:

Example

```
ol {
  counter-reset: section;
  list-style-type: none;
}
```

```
li::before {
  counter-increment: section;
  content: counters(section,".") " ";
}
```

CSS Counter Properties

Property	Description
content	Used with the ::before and ::after pseudo-elements, to insert generated content
counter-increment	Increments one or more counter values
counter-reset	Creates or resets one or more counters

CSS Website Layout

Website Layout

A website is often divided into headers, menus, content and a footer:

There are tons of different layout designs to choose from. However, the structure above, is one of the most common, and we will take a closer look at it in this tutorial.

Header

A header is usually located at the top of the website (or right below a top navigation menu). It often contains a logo or the website name:

Example

```
.header {
  background-color: #F1F1F1;
  text-align: center;
  padding: 20px;
}
```

Result

Header

Navigation Bar

A navigation bar contains a list of links to help visitors navigating through your website:

Example

```
/* The navbar container */
.topnav {
  overflow: hidden;
  background-color: #333;
}

/* Navbar links */
.topnav a {
  float: left;
  display: block;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

/* Links - change color on hover */
.topnav a:hover {
  background-color: #ddd;
  color: black;
}
```

Result

Content

The layout in this section, often depends on the target users. The most common layout is one (or combining them) of the following:

- **1-column** (often used for mobile browsers)
- **2-column** (often used for tablets and laptops)
- **3-column layout** (only used for desktops)

1-column:

2-column:

3-column:

We will create a 3-column layout, and change it to a 1-column layout on smaller screens:

Example

```
/* Create three equal columns that floats next to each other */
.column {
  float: left;
  width: 33.33%;
}

/* Clear floats after the columns */
.row:after {
  content: "";
  display: table;
  clear: both;
}

/* Responsive layout - makes the three columns stack on top of each other instead of next to each other on smaller screens (600px wide or less) */
@media screen and (max-width: 600px) {
  .column {
    width: 100%;
  }
}
```

Result

Column

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque ultricies, eget elementum magna tristique.

Column

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque ultricies, eget elementum magna tristique.

Column

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque ultricies, eget elementum magna tristique.

Tip: To create a 2-column layout, change the width to 50%. To create a 4-column layout, use 25%, etc.

Tip: Do you wonder how the @media rule works? Read more about it in our CSS Media Queries chapter.

Tip: A more modern way of creating column layouts, is to use CSS Flexbox. However, it is not supported in Internet Explorer 10 and earlier versions. If you require IE6-10 support, use floats (as shown above).

To learn more about the Flexible Box Layout Module, read our CSS Flexbox chapter.

Unequal Columns

The main content is the biggest and the most important part of your site.

It is common with **unequal** column widths, so that most of the space is reserved for the main content. The side content (if any) is often used as an alternative navigation or to specify information relevant to the main content. Change the widths as you like, only remember that it should add up to 100% in total:

Example

```
.column {
  float: left;
}

/* Left and right column */
.column.side {
  width: 25%;
}

/* Middle column */
.column.middle {
  width: 50%;
}

/* Responsive layout - makes the three columns stack on top of each other instead of next to each other */
@media screen and (max-width: 600px) {
  .column.side, .column.middle {
```

```
    width: 100%;
  }
}
```

Result

Side

Lorem ipsum dolor sit amet, consectetur adipiscing elit...

Main Content

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet pretium urna. Vivamus venenatis velit nec neque ultricies, eget elementum magna tristique. Quisque vehicula, risus eget aliquam placerat, purus leo tincidunt eros, eget luctus quam orci in velit. Praesent scelerisque tortor sed accumsan convallis.

Side

Lorem ipsum dolor sit amet, consectetur adipiscing elit...

Footer

The footer is placed at the bottom of your page. It often contains information like copyright and contact info:

Example

```
.footer {
  background-color: #F1F1F1;
  text-align: center;
  padding: 10px;
}
```

Result

Footer

Responsive Website Layout

By using some of the CSS code above, we have created a responsive website layout, which varies between two columns and full-width columns depending on screen width:

CSS Units

CSS Units

CSS has several different units for expressing a length.

Many CSS properties take "length" values, such as `width`, `margin`, `padding`, `font-size`, etc.

Length is a number followed by a length unit, such as `10px`, `2em`, etc.

Example

Set different length values, using px (pixels):

```
h1 {
  font-size: 60px;
}

p {
  font-size: 25px;
  line-height: 50px;
}
```

Note: A whitespace cannot appear between the number and the unit. However, if the value is `0`, the unit can be omitted.

For some CSS properties, negative lengths are allowed.

There are two types of length units: **absolute** and **relative**.

Absolute Lengths

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout.

Unit	Description
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

* Pixels (px) are relative to the viewing device. For low-dpi devices, 1px is one device pixel (dot) of the display. For printers and high resolution screens 1px implies multiple device pixels.

Relative Lengths

Relative length units specify a length relative to another length property. Relative length units scales better between different rendering mediums.

Unit	Description
em	Relative to the font-size of the element (2em means 2 times the size of the current font)
ex	Relative to the x-height of the current font (rarely used)
ch	Relative to width of the "0" (zero)
rem	Relative to font-size of the root element

vw	Relative to 1% of the width of the viewport*
vh	Relative to 1% of the height of the viewport*
vmin	Relative to 1% of viewport's* smaller dimension
vmax	Relative to 1% of viewport's* larger dimension
%	Relative to the parent element

Tip: The em and rem units are practical in creating perfectly scalable layout!
* Viewport = the browser window size. If the viewport is 50cm wide, 1vw = 0.5cm.

Browser Support

The numbers in the table specify the first browser version that fully supports the length unit.

Length Unit					
em, ex, %, px, cm, mm, in, pt, pc	1.0	3.0	1.0	1.0	3.5
ch	27.0	9.0	1.0	7.0	20.0
rem	4.0	9.0	3.6	4.1	11.6
vh, vw	20.0	9.0	19.0	6.0	20.0
vmin	20.0	12.0	19.0	6.0	20.0
vmax	26.0	16.0	19.0	7.0	20.0

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!

Note: Specificity is a common reason why your CSS-rules don't apply to some elements, although you think they should.

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.

How to Calculate Specificity?

Memorize how to calculate specificity!

Start at 0, add 1000 for style attribute, add 100 for each ID, add 10 for each attribute, class or pseudo-class, add 1 for each element name or pseudo-element.

Consider these three code fragments:

Example

```
A: h1
B: #content h1
C: <div id="content"><h1 style="color: #ffffff">Heading</h1></div>
```

The specificity of A is 1 (one element)
The specificity of B is 101 (one ID reference and one element)
The specificity of C is 1000 (inline styling)

Since 1 < 101 < 1000, the third rule (C) has a greater level of specificity, and therefore will be applied.

Specificity Rules

Equal specificity: the latest rule counts - If the same rule is written twice into the external style sheet, then the lower rule in the style sheet is closer to the element to be styled, and therefore will be applied:

Example

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

the latter rule is always applied.

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

Example

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

the first rule is more specific than the other two, and will be applied.

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

Example

From external CSS file:

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

In HTML file:

```
<style>
#content h1 {
  background-color: yellow;
}
</style>
```

the latter rule will be applied.

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

Example

```
.intro {background-color: yellow;}
h1 {background-color: red;}
```

The universal selector and inherited values have a specificity of 0 **- *, body * and similar have a zero specificity. Inherited values also have a specificity of 0. CSS The !important Rule**

What is !important?

The `!important` rule in CSS is used to add more importance to a property/value than normal.

In fact, if you use the `!important` rule, it will override ALL previous styling rules for that specific property on that element!

Let us look at an example:

Example

```
#myid {
  background-color: blue;
}

.myclass {
  background-color: gray;
}

p {
  background-color: red !important;
}
```

Example Explained

In the example above. all three paragraphs will get a red background color, even though the ID selector and the class selector has a higher specificity. The `!important` rule overrides the `background-color` property in both cases.

Important About !important

The only way to override an `!important` rule is to include another `!important` rule on a declaration with the same (or higher) specificity in the source code - and here the problem starts! This makes the CSS code confusing and the debugging will be hard, especially if you have a large style sheet!

Here we have created a simple example. It is not very clear, when you look at the CSS source code, which color is considered most important:

Example

```
#myid {
  background-color: blue !important;
}
```

```
.myclass {
  background-color: gray !important;
}

p {
  background-color: red !important;
}
```

Tip: It is good to know about the `!important` rule, you might see it in some CSS source code. However, do not use it unless you absolutely have to.

Maybe One or Two Fair Uses of !important

One way to use `!important` is if you have to override a style that cannot be overridden in any other way. This could be if you are working on a Content Management System (CMS) and cannot edit the CSS code. Then you can set some custom styles to override some of the CMS styles.

Another way to use `!important` is: Assume you want a special look for all buttons on a page. Here, buttons are styled with a gray background color, white text, and some padding and border:

Example

```
.button {
  background-color: #8c8c8c;
  color: white;
  padding: 5px;
  border: 1px solid black;
}
```

The look of a button can sometimes change if we put it inside another element with higher specificity, and the properties gets in conflict. Here is an example of this:

Example

```
.button {
  background-color: #8c8c8c;
  color: white;
  padding: 5px;
  border: 1px solid black;
```

```
}

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

To "force" all buttons to have the same look, no matter what, we can add the `!important` rule to the properties of the button, like this:

Example

```
.button {
  background-color: #8c8c8c !important;
  color: white !important;
  padding: 5px !important;
  border: 1px solid black !important;
}

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