# **HTML5 Semantic Elements**

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of **non-semantic** elements: <div> and <span> - Tells nothing about its content.

Examples of **semantic** elements: <form>, <table>, and <article> - Clearly defines its content.

<article> Defines an article

<aside> Defines content aside from the page content

<details> Defines additional details that the user can view or hide

<figcaption> Defines a caption for a <figure> element

<figure> Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.

<footer> Defines a footer for a document or section

<header> Specifies a header for a document or section

<main> Specifies the main content of a document

<mark> Defines marked/highlighted text

<nav> Defines navigation links

<section> Defines a section in a document

<summary> Defines a visible heading for a <details> element

<time> Defines a date/time

# **CSS Flexbox**

Adding display: flex to an element turns it into a flex container. This makes it possible to align any children of that element into rows or columns. You do this by adding the flex-direction property to the parent item and setting it to row or column. Creating a row will align the children horizontally, and creating a column will align the children vertically.

<style>

#box-container {

display: flex;

height: 500px;

flex-direction: row;

}

#box-1 {

background-color: dodgerblue;

width: 50%;

height: 50%;

}

#box-2 {

background-color: orangered;

width: 50%;

height: 50%;

}

</style>

<div id="box-container">

<div id="box-1"></div>

<div id="box-2"></div>

</div>



<style>

#box-container {

display: flex;

height: 500px;

flex-direction: column;

}

#box-1 {

background-color: dodgerblue;

width: 50%;

height: 50%;

}

#box-2 {

background-color: orangered;

width: 50%;

height: 50%;

}

</style>

<div id="box-container">

<div id="box-1"></div>

<div id="box-2"></div>

</div>



There are several options for how to space the flex items along the line that is the main axis. One of the most commonly used is **justify-content: center;,** which aligns all the flex items to the center inside the flex container. Others options include:

* flex-start: aligns items to the start of the flex container. For a row, this pushes the items to the left of the container. For a column, this pushes the items to the top of the container.
* flex-end: aligns items to the end of the flex container. For a row, this pushes the items to the right of the container. For a column, this pushes the items to the bottom of the container.
* space-between: aligns items to the center of the main axis, with extra space placed between the items. The first and last items are pushed to the very edge of the flex container. For example, in a row the first item is against the left side of the container, the last item is against the right side of the container, then the other items between them are spaced evenly.
* space-around: similar to space-betweenbut the first and last items are not locked to the edges of the container, the space is distributed around all the items

**align-items:** property to align flex items along the cross axis. For a row, it tells CSS how to push the items in the entire row up or down within the container. And for a column, how to push all the items left or right within the container.

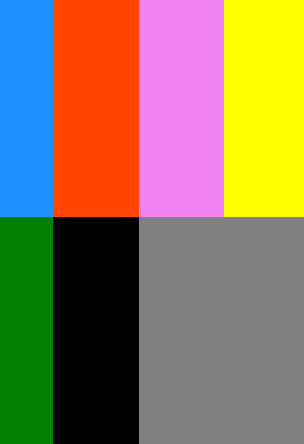
The different values available for align-items include:

* flex-start: aligns items to the start of the flex container. For rows, this aligns items to the top of the container. For columns, this aligns items to the left of the container.
* flex-end: aligns items to the end of the flex container. For rows, this aligns items to the bottom of the container. For columns, this aligns items to the right of the container.
* center: align items to the center. For rows, this vertically aligns items (equal space above and below the items). For columns, this horizontally aligns them (equal space to the left and right of the items).
* stretch: stretch the items to fill the flex container. For example, rows items are stretched to fill the flex container top-to-bottom.
* baseline: align items to their baselines. Baseline is a text concept, think of it as the line that the letters sit on.

**flex-wrap:** property, it tells CSS to wrap items. This means extra items move into a new row or column. The break point of where the wrapping happens depends on the size of the items and the size of the container.

CSS also has options for the direction of the wrap:

* nowrap: this is the default setting, and does not wrap items.
* wrap: wraps items from left-to-right if they are in a row, or top-to-bottom if they are in a column.
* wrap-reverse: wraps items from bottom-to-top if they are in a row, or right-to-left if they are in a column.



**flex-shrink:** property. When it's used, it allows an item to shrink if the flex container is too small. Items shrink when the width of the parent container is smaller than the combined widths of all the flex items within it.

The flex-shrink property takes numbers as values. The higher the number, the more it will shrink compared to the other items in the container. For example, if one item has a flex-shrinkvalue of 1 and the other has a flex-shrink value of 3, the one with the value of 3 will shrink three times as much as the other.



**flex-grow:** property is opposite of flex-shrink is the . Recall that flex-shrink controls the size of the items when the container shrinks. The flex-growproperty controls the size of items when the parent container expands.

Using a similar example from the last challenge, if one item has a flex-growvalue of 1 and the other has a flex-growvalue of 3, the one with the value of 3 will grow three times as much as the other.

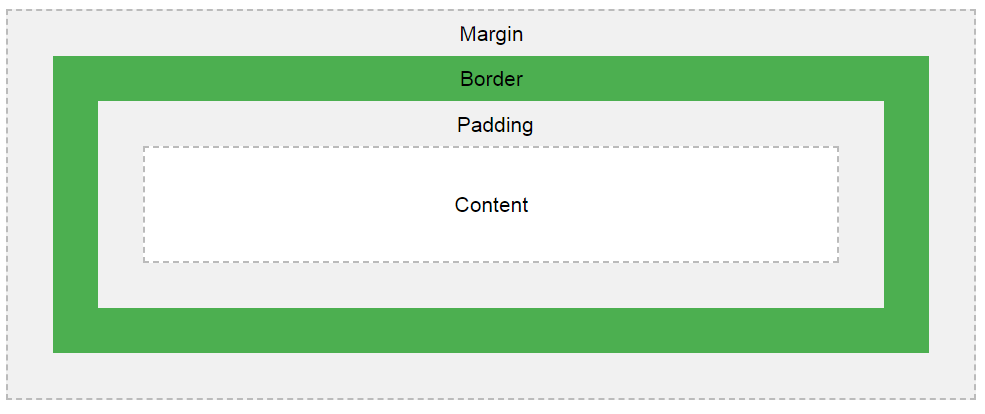
**flex-basis:** property specifies the initial size of the item before CSS makes adjustments with flex-shrinkor flex-grow.

The units used by the flex-basisproperty are the same as other size properties (px, em, %, etc.). The value autosizes items based on the content.

**align-self**: This property allows you to adjust each item's alignment individually, instead of setting them all at once. This is useful since other common adjustment techniques using the CSS properties float, clear, and vertical-aligndo not work on flex items.

align-self accepts the same values as align-itemsand will override any value set by the align-items property

# **CSS Box Model**



* **Content** - The content of the box, where text and images appear
* **Padding** - Clears an area around the content. The padding is transparent
* **Border** - A border that goes around the padding and content
* **Margin** - Clears an area outside the border. The margin is transparent

**The total width of an element should be calculated like this**:

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

div {  
  width: 320px;  
  padding: 10px;  
  border: 5px solid gray;  
  margin: 0; }

Here is the calculation: 320px (width)  
+ 20px (left + right padding)  
+ 10px (left + right border)  
+ 0px (left + right margin)  
**= 350px**

**CSS:position values**

1. Static

position: static is the default value. Whether we declare it or not, elements are positioned in normal order on the webpage. Let’s give an example:

First, we define our HTML structure:

<body>

<div class="box-orange"></div>

<div class="box-blue"></div>

</body>

Then, we create 2 boxes and define their positions:

.box-orange { // without any position declaration

background: orange;

height: 100px;

width: 100px;

}

.box-blue {

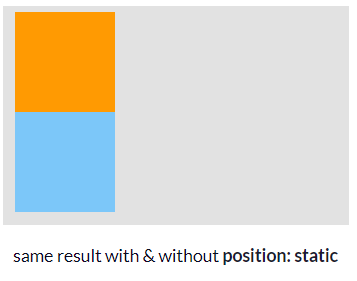
background: lightskyblue;

height: 100px;

width: 100px;

position: static; // Declared as static

}



Div’s are **block elements** by default, and that’s why they are not on the same line.

### **2. Relative**

position: relative: An element’s new position relative to its normal position.

Starting with position: relative and for all **non-static**positionvalues, we are able to change an element’s **default**position. But only defining position: relative is not enough, we also need to set the element’s coordinates with **helper properties.**

Let’s move the orange box next to the blue one.

.box-orange {

position: relative; // We can now move the element

background: orange;

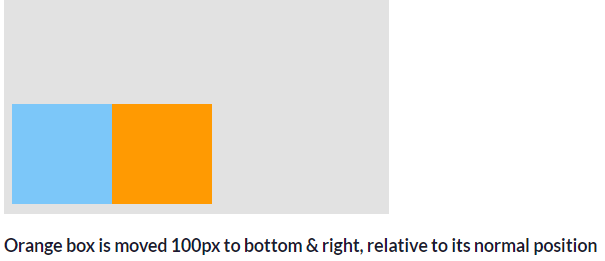
width: 100px;

height: 100px;

top: 100px; // 100px from top relative to its old position

left: 100px; // 100px from left

}



*NOTE: Using****position: relative****for an element, doesn’t affect other elements’ positions.*

### **3. Absolute**

In position: relative, the element is positioned **relative to itself.** However, an **absolute**positioned element is **relative to its parent**.

An element with position: absolute is removed from the normal document flow. It is positioned automatically to the starting point (**top-left corner)** of its parent element. If it doesn’t have any parent elements, then the initial**document <html>** will be its parent.

Since position: absolute removes the element from the document flow, other elements **are affected**and behave as the element is removed completely from the webpage.

Let’s add a **container** as parent element:

<body>

<div class="container">

<div class="box-orange"></div>

<div class="box-blue"></div>

</div>

</body>

.box-orange {

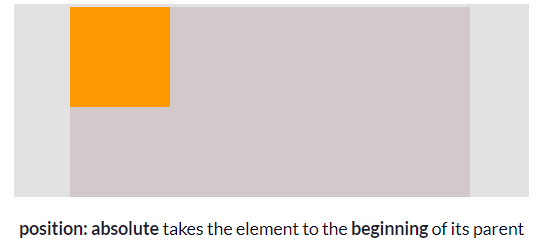
position: absolute;

background: orange;

width: 100px;

height: 100px;

}



Now it looks like the blue box has disappeared, but it hasn’t. The blue box behaves like the orange box is removed, so it shifts up to the orange box’s place.

The coordinates of an **absolute** positioned element are **relative to its parent** if the parent also has a **non-static position.**Otherwise, helper properties position the element relative to the **initial <html>.**

### **4. Fixed**

Like position: absolute, fixed positioned elements are also removed from the normal document flow. The differences are:

* They are **only relative to the <html> document,**not any other parents.
* They are **not affected by scrolling**.

Here in the example, I change the orange box’s position to **fixed**, and this time it is relative 5px to the right of the **<html>**, not its**parent (container).**

As we can see, scrolling the page doesn’t affect the **fixed** positioned box. It is not relative to its parent (container) anymore.

### **5. Sticky**

position: sticky can be explained as a mix of position: relative and position: fixed.

It behaves until a declared point like position: relative, after that it changes its behavior to position: fixed .

# HTML 5 Features

Web languages need regular upgrades in order to stay current and solve new problems faced by web developers. HTML5 is the latest version of HTML. Below are some HTML5 features you will encounter as you learn with Codecademy.

## VIDEO

The video element allows you to easily stream video from a website.

<video width="450px" height="350px" controls>

<source src="video-url.mp4" type="video/mp4">

</video>

In the HTML above, width and height set the dimensions for the videoelement. The controls attribute creates playback buttons such as “Play” and “Pause”. The source src tag provides the URL where the video is hosted and type specifies the video’s type, in this case, “video/mp4”.

## FIGURE

Figure elements can be used to display visual content, such as photos, illustrations, diagrams or code snippets.

<figure class="gallery-item">

<img src="image-1.png">

</figure>

<figure class="gallery-item">

<img src="image-2.png">

</figure>

In the example code above, figure elements have the class “gallery-item”, and each contains an img element.

## SECTION

Section elements, like divs, can be used to organize webpage content into thematic groups.

<section class="contact-form">

<h2>Contact Us</h2>

<form>

...

</form>

</section>

Above, a section element is used to organize h2 and form elements for a website’s “Contact Us” feature.

## NAV

The nav element is used for the part of a website that links to other pages on the site. The links can be organized a number of ways. Below, the links are displayed within paragraph elements. An unordered list could also be used.

<nav>

<p><a href="login.html">Log In</a></p>

<p><a href="signup.html">Sign Up</a></p>

<p><a href="contact.html">Contact Us</a></p>

</nav>

## HEADER

The header element can be used to group together introductory elements on a website, such as a company logo, navigation items, and sometimes, a search form.

<header>

<img src="company-logo.png">

<nav>

<p><a href="login.html">Log In</a></p>

<p><a href="signup.html">Sign Up</a></p>

<p><a href="contact.html">Contact Us</a></p>

</nav>

</header>

Above, the header element encloses the img and nav.

## FOOTER

The footer element is typically found at the bottom or foot of a webpage. It can contain copyright information, links to social media and additional site navigation items.

<footer>

<p>&copy; Acme Granola Corporation 2016<p>

<div class="social">

<a href="#"><img src="instagram-icon.png"></a>

<a href="#"><img src="facebook-icon.png"></a>

<a href="#"><img src="twitter-icon.png"></a>

</div>

</footer>

Above, between <footer> and </footer>, copyright information is contained in the p element, and social media links are contained within the div with class “social”.

# CSS3 Features

CSS3 is the latest version of CSS and contains a number of exciting, new features that make it easier for web developers to create great styling for websites.

## REM VALUES

You will encounter rem values as you learn about the CSS font-size property, and other CSS properties that specify element size. In the case of the font-size property, a rem value displays a font-size relative to the font-size of the root element, called simply html.

For example, consider the following CSS:

html {

font-size: 20px;

}

p {

font-size: 0.75rem;

}

In the code, font-size is set to 20px for the html selector. The p selector has a font-size of 0.75rem, which displays a size 3/4 that of html, or 15px.

## FLEXBOX

The CSS3 flexbox feature makes it much easier for web developers to arrange HTML elements vertically or horizontally. Website layouts designed with flexbox can respond to users with various screen widths, including mobile devices. To access the feature in CSS, the display property must be set to flex, as seen below:

.main {

display: flex;

}

HTML elements that are children of an element with the “main” class are now flex items and can be arranged using flexbox properties. Here are two such properties:

* flex-wrap: arranges flex items into a single line or wraps them across multiple lines of a webpage layout.
* justify-content: can orient flex items in a number of different positions on a webpage, including the center.

## BACKGROUND-IMAGE

The CSS3 background-image property is used to set a background image for an HTML element.

For example, consider the following CSS:

body {

background-image: url("http://image-gallery.io/mountain-scene.png");

}

## BACKGROUND-SIZE

The CSS background-size property controls the size of an HTML element’s background image. Used along with the background-image property, background-size values control how a background image is proportioned and scaled. One such value is cover as seen in the following CSS:

body {

background-image: url("http://image-gallery.io/mountain-scene.png");

background-size: cover;

}

Here, the image covers the entire HTML body element.

# **CSS Preprocessors - Sass vs LESS**

## What is a CSS Preprocessor?#

Writing CSS can become quite repetitive and little tasks such as having to look up hex color values, closing your tags, etc. can become time-consuming. And so that is where a preprocessor comes into play. A CSS preprocessor is basically a **scripting language that extends CSS** and then **compiles it into regular CSS**.

**Advantages to Using a Preprocessor**

1. Cleaner code with reusable pieces and variables
2. Saves you time
3. Easier to maintain code with snippets and libraries
4. Calculations and logic
5. More organized and easy to setup

## Sass Vs LESS

Sass and LESS are both very powerful CSS extensions. You can think of them as more of a programming language designed to make CSS more maintainable, themeable, and extendable. Both Sass and LESS are backward compatible so you can easily convert your existing CSS files just by renaming the .css file extension to .less or .scss, respectively. LESS is JavaScript based and Sass is Ruby based.

**Block Tags and Inline Tags**

## Block Elements

Block elements appear on the screen as if they have a line break before and after them. For example, the <p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <ul>, <ol>, <dl>, <pre>, <hr />, <blockquote>, and <address> elements are all block level elements. They all start on their own new line, and anything that follows them appears on its own new line.

## Inline Elements

Inline elements, on the other hand, can appear within sentences and do not have to appear on a new line of their own. The <b>, <i>, <u>, <em>, <strong>, <sup>, <sub>, <big>, <small>, <li>, <ins>, <del>, <code>, <cite>, <dfn>, <kbd>, and <var> elements are all inline elements.

# **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 ::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.

## Pseudo-elements and CSS Classes

Pseudo-elements can be combined with CSS classes:

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

## Multiple Pseudo-elements

Several pseudo-elements can also be combined.

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

### **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:

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:

h1::after {

content: url(smiley.gif);}

## CSS - The ::selection Pseudo-element

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

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

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

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

**Open page in new tab:**

function openInNewTab(url) {

var win = window.open(url, '\_blank');

win.focus();

}

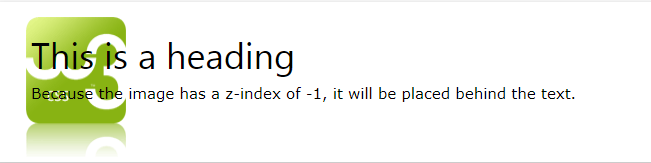
<div onclick="openInNewTab('www.test.com');">Something To Click On</div>

## 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:



# Pseudo-classes

A [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) **pseudo-class** is a keyword added to a selector that specifies a special state of the selected element(s). For example, [:hover](https://developer.mozilla.org/en-US/docs/Web/CSS/:hover) can be used to change a button's color when the user's pointer hovers over it.

/\* Any button over which the user's pointer is hovering \*/

button:hover {

color: blue;

}

**Example:**

# :focus

The **:focus** [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) [pseudo-class](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes) represents an element (such as a form input) that has received focus. It is generally triggered when the user clicks or taps on an element or selects it with the keyboard's "tab" key.

/\* Selects any <input> when focused \*/

input:focus { color: red;}