Code School Cross Country CSS Notes

Level 1 – Fundamentals

***CSS Rule Locations***

There are basically three locations where CSS definitions can be placed. These are:

* Inline;
* In the Head Element
* External Style Sheet

***CSS Rule Associations***

An external style sheet is preferred for each HTML document. A CSS rule can be associated with an HTML element using three different methods.

These are:

* Element;
* ID
* Class

These are demonstrated below:

<h1 class = “mark” id = “header”> mark the koala </h1>

And then the classes can be defined as:

h1 {

color: #aba4ac

}

.mark {

font-size: 15px;

margin-bottom: 10px;

}

#header {

margin-top: 50px;

}

There can be compound selectors. This rule is applied to ids of header which are defined inside an h1 element.

h1#header {

color: #aba4ac

}

***CSS Rule Precedence***

External is overridden by head which is overridden by inline. If there are two classes such as the classes defined below:

.intro {

color: #aba4ac

}

.intro {

color: #aba4ac

}

The second class will win. If there are no conflicts, then the result is the **union** of the two rules.

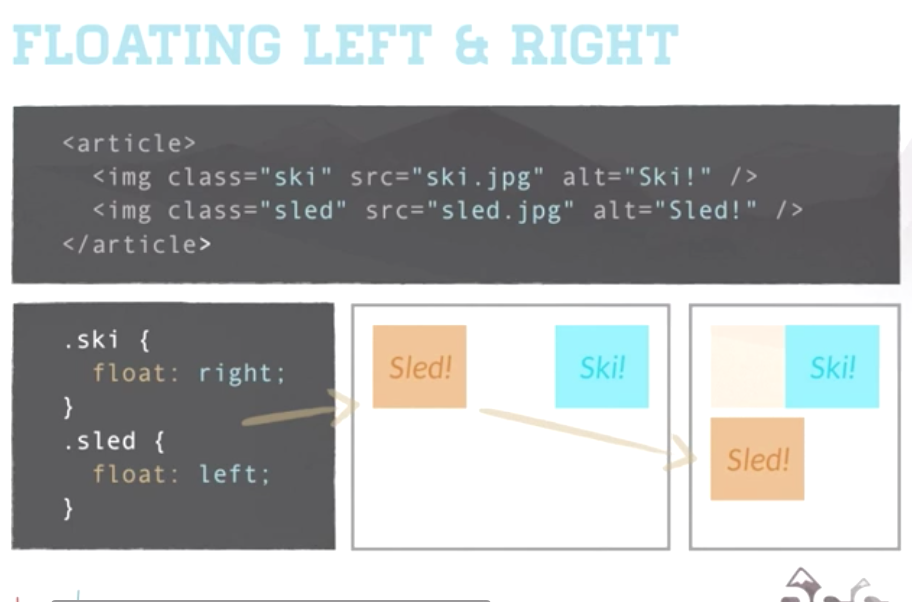
***Floats***

Floats remove elements from the document flow and move them to a specified edge. Other content within the parent element will wrap around the float. Floats can be “left” / “right” / “none” We use “none” to remove a float.

In regards to three elements (i.e. say images floated next to each other) If the browser’s width is reduced sufficiently, then the rightmost element will be moved down and will be flush with the left of the browser. But this is not the case of the leftmost element is quite tall. Please see below:



In regards to the following. The sled is defined in the HTML ***after*** the ski image. This means that when the browser is shrunk, then the sled image will break first. Even though the sled is ***displayed*** before the ski image – because of the CSS settings. So, when the browser is shrunk, the sled will appear on the left on a new line and the ski will appear right on the first line.



When multiple images are stacked up against each other as in the following. As ski is defined before sled the ski is at the far right.

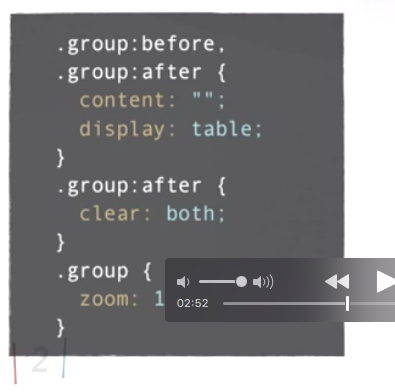


Level 2 – Clear Carving

***Clearing floats***

The following is from ( <https://www.w3schools.com/css/css_float.asp> ) If an element is taller than the element containing it, and is floated, it will overflow outside of its container. Have a look at this example: <https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clearfix> In this example, we have a <div> and inside this div is some text and an image. The div has a green border and the image is floated to the right. The image is larger than the text. So, it extends outside the div. (probably need to have a look at the url). To fix this a property of “overflow: auto” was added to the class. But a more modern hack is generally used.

Here is the CSS (??) “The “ClearFix” The Code School explanations are too difficult to understand.

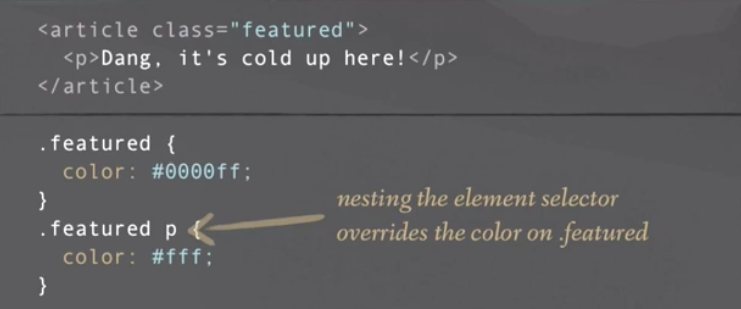


***Inheritance and specificity***

The child inherits properties of the parent. For example. The color of the p element below will be the same as its container (the article class).



If you need to override the color for the p element. Use a nested selector. See below:



Note with the element selector we are using:

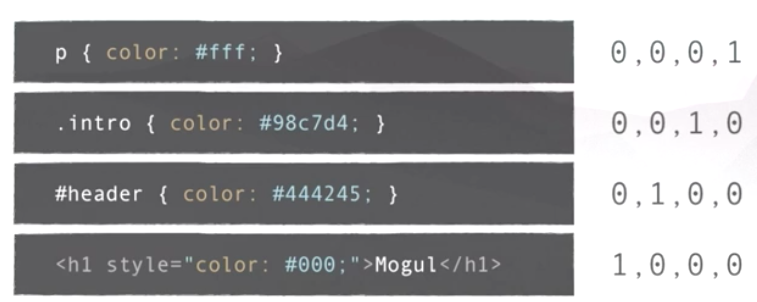
.featured p { }

***Dealing with specificity***

Which has priority: the class, the id or the element. This can be represented by a comma delimited list of 4 digits: 0, 0, 0, 0.



This is how the above image is implemented:



With the above, the inline style (the last entry) would have priority. The “! important; “ value will win over everything.

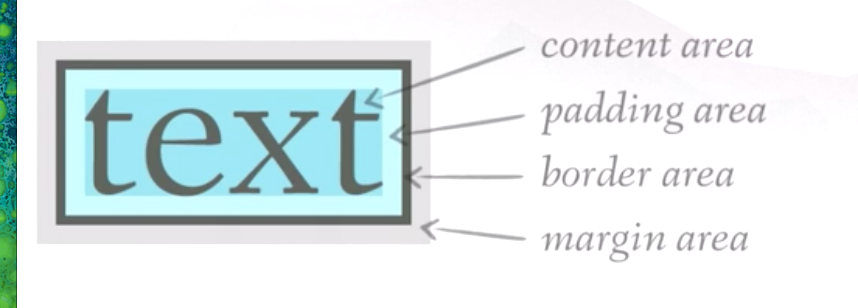
Here is a more complex example:



So, in the first entry above, we have two class selectors and 1 element sector. In the second row, we have 2 class selectors and 2 element selectors.

Level 3 – Box Bindings

The following shows which box attributes have which names.



***Calculating width***

The total calculates box width is the content width + combined padding width + border width.

***The overflow model***

This is what happens if the content is wider than the box. So, if you have an image that is 400 px but the underlying box is 300 px.

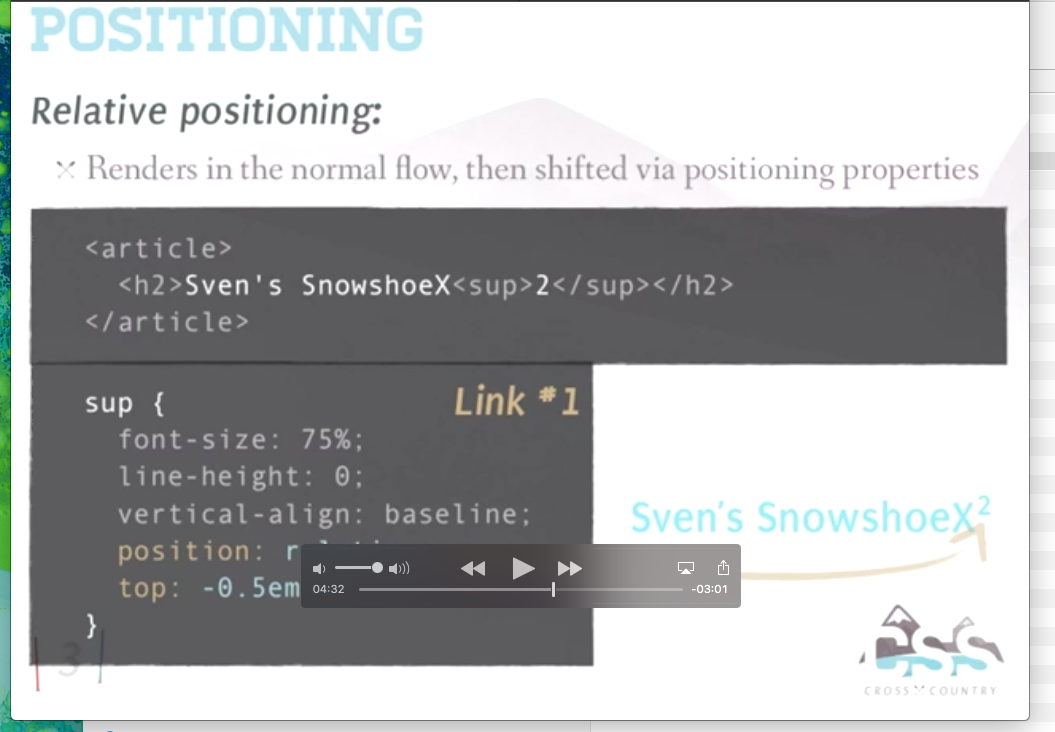
The overflow property can take four different values. These are: visible, auto, hidden, scroll.

The **visible** is the default value and this allows content to extend beyond container boundaries. **Auto** adds a scrollbar as needed when the content overflows. The **hidden** hides content that extends beyond the container. **Scroll** adds a scrollbar at all times.

***Positioning***

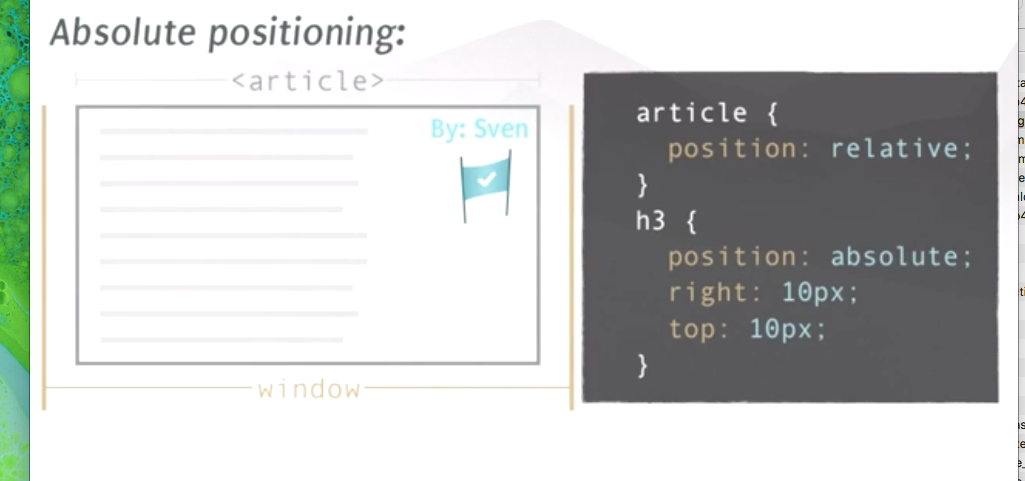
The position property can take four values. These are: *static, relative, absolute and fixed*. Anything that’s on the page as part of the normal flow is considered static. Using a ***value other than static*** causes an object to become a ***positioned element.*** Positioned elements may use the top, left, bottom and right properties.

Here is an example. So, it has a position: relative, so it is not static and therefore is a positioned element. You can’t see it in the image below, but the position is “relative”



The **relative** element renders in the normal flow and is then shifted via positioning properties. See image above.

The **absolute** positioning takes the element out of the normal flow for manual positioning. The absolute position automatically is scoped to the window **unless it falls within an element that is positioned.** Such that the positioning is scoped to the parent window, the parent should be set with a property of position: relative.



Fixed positioning fixes an element to a specific place in the window where it will stay, **regardless of scrolling.**

***Z Indexing***

The image which occurs second in the html will occur above the first. But higher Z index values occur above lower Z index values.

Level 4 – Grooming your code

***Staying Dry***

Don't use multiple CSS selectors that say the same thing.

***Selectors at a higher HTML level***

For example, if you have an h1 and a P element that have a font-family of Arial, then you do not want to declare Aria in the P and in the H1 elements. Instead, you just declare it in the body element. Or, if this is too general, well then you just wrap things in a <DIV> and then use a class that is attributed to the DIV.

***Aggregating Selectors***

Another way of staying DRY is that if you have a say a <p> element, <h6> element and a “.ski\_lift” class, instead of declaring them all separately with say the same font size you can just put them all into a comma delimited list, like the following:

p, .ski\_lift, h6 { color: #333 }

***Selector Abstraction***

If you have two buttons and they have basically the same style but they have different background colors, well you can do the following. Note that order matters. Then your html, would be the following:

<input type = “submit” class=”button submit” ….>

***Simplifying Margin Declarations***

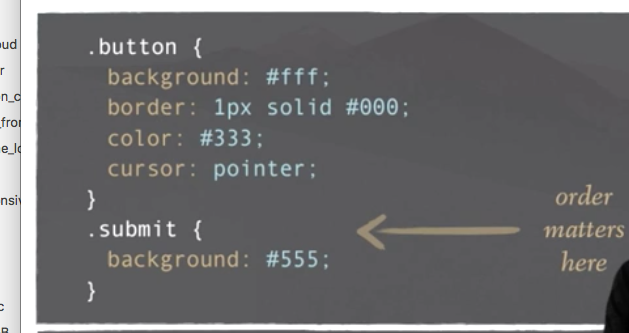
You could write:

.ski\_poles {margin-top: 15px; margin-left: 20px …..}

This can be simplified as:

.ski\_poles {margin: 15px 10px 0 20px}

*See the following table for short cuts:*



You can also use the following CSS short hands:

|  |  |
| --- | --- |
| Property | Value |
| font | *Style weight size/line-height family* |
| Italic bold 16px/18px sans-serif |
| Background | *Color image repeat x-pos y-pox* |
| #000 url(image.jpg) no-repeat center top; |
| List-style | Style position image |
| Disc inside none |
| Border | Width style color |
| 3px solid #ccc; |
| Margin or padding | 0 10x 0 10px / 0 10px 0 / 0 10px; |
| Top right bottom left // top righ & left bottom // top & bottom right & left |

***Display types***

For the property “display” there are none, block, inline, inline-block.

**Block elements** stretch the full width of their container and behave as though there is a line break before before and after the element. The full box model can be implemented. These are tags such as: <div> <p> <ul> <ol> <li>

You center these with: margin: 0 auto;

**Inline elements** are typically found within block-level elements and they only take up the space of the content inside. They do not generate a line break before or after the content. These are tags such as <span> <a> <em> <img> and <strong>.

**Inline block** have the same flow as an inline element but behave the same as a block element.



You can use the full box model with inline-blocks.

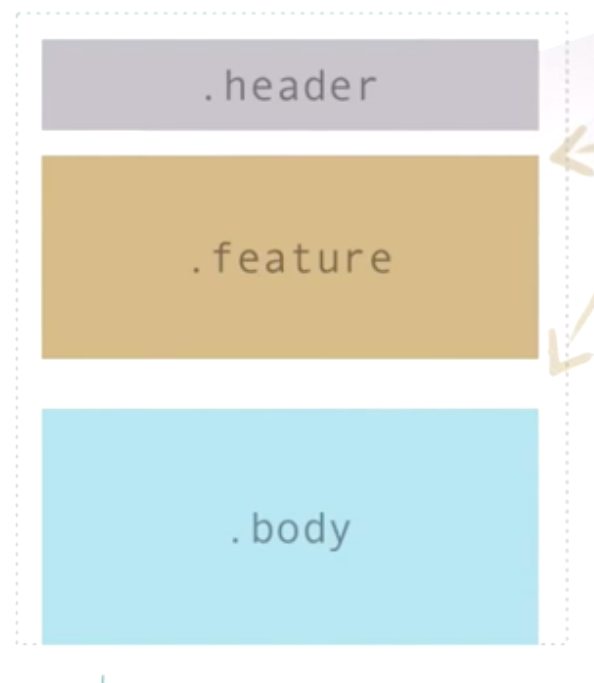
***Centering***

Depending on the display type (see above), there are different ways of centering the element.

For a **block** element use “auto” . Such as: margin: 0 auto;

For inline and inline-block elements use “text-align:center”

Level 5 – Protecting your layout



If you apply top and bottom margins to the feature section above and then delete the feature section then your header and body will be flush up against each other. So, what you might do is incorporate the following CSS:



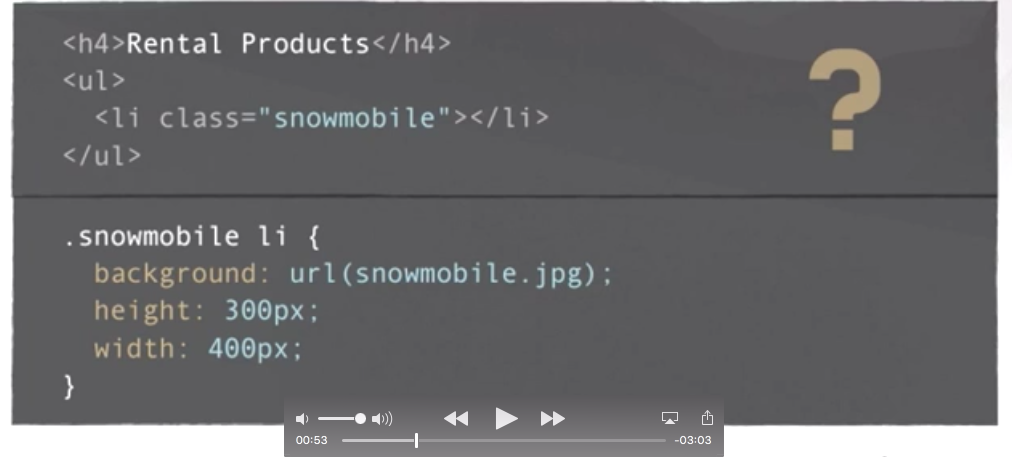
You might expect that the distance between header and feature will be (40 + 20) = 60 px. But what actually happens, due to declaring margins on subsequent elements, is that due to a process called **“Margin Collapsing”.**

What happens above is that the **larger** of the two margins for the header and the feature will be used. This means that 40px will be used as a margin.

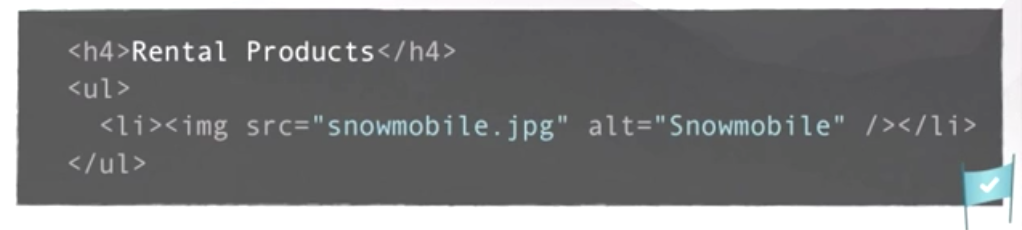
Collapsing margins will ***not occur*** when one or more block elements have: Padding or borders, relative or absolute positioning or a float left or right.

Level 6 – Image Issues

Sometimes it is better to incorporate an image as an inline element rather than CSS. This is because the HTML is accessible by code and therefore the image can be dynamically changed. The following shows the CSS way.



And the following shows it as an inline image:



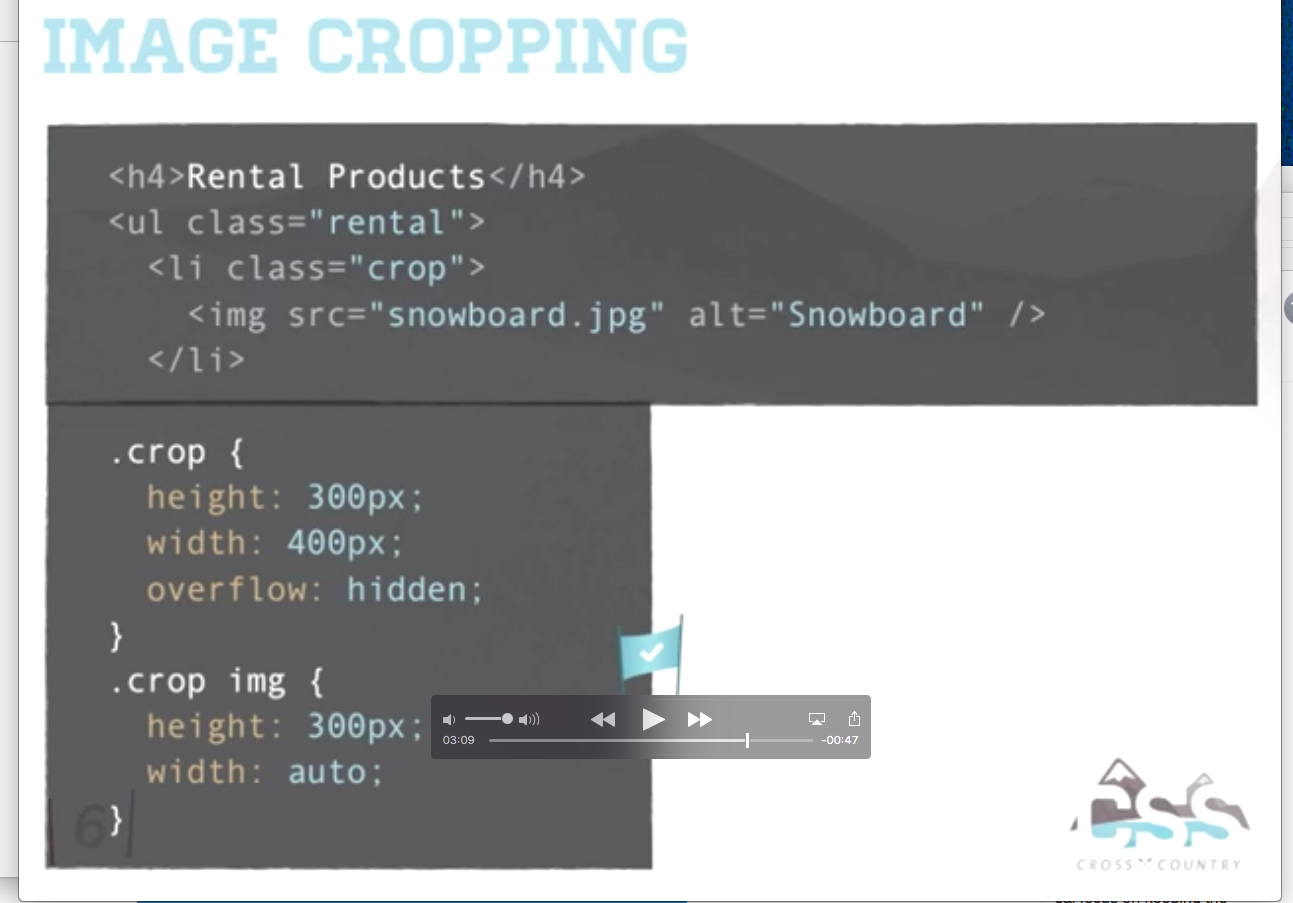
But you shouldn’t always use an inline image. For example if an image is part of the layout (such as a divider for H1) the image should be defined as a background image.

***Image Cropping – the overflow crop method***

When the layout gets changed the image might distort as the size of the image may have been statically defined.



The above will crop the image if its space gets too small. It will not hide the image but the image proportions will not change. But this is not perfect as we really do not want part of the image to be hidden. Instead we would like the image to be scaled to better fit its container. This is shown below:



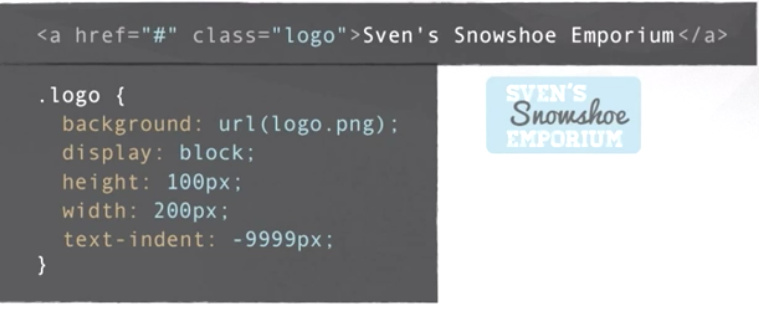
When the image is portrait the width is set to auto; when the image is landscape the height is set to auto. But we can’t really account for mixed images.

The author says that there is no perfect solution. The author talks about resizing images server side.

Level 7 – Image Issues

***Image replacement***

Use the following pattern to include a text element for a screen reader:

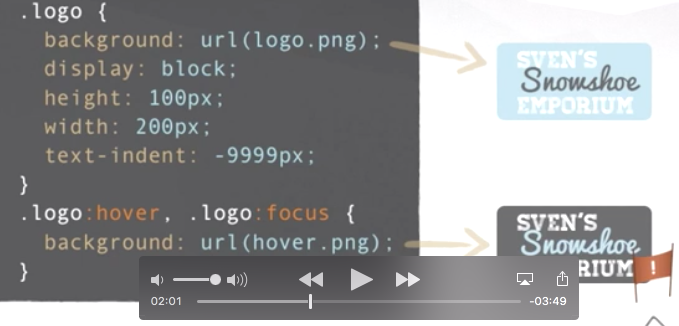


The above works as the screen reader can read “Sven’s Snowshoe Emporium” but this text not displayed due to the “text-indent” property in the CSS.

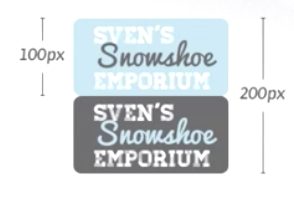
***Sprites.***

The following attempts to display two images. One image for the normal state and the other image for the hover / focus state.

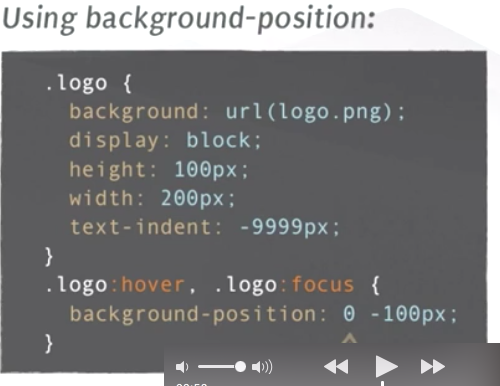
The following works fine. But when the event fires, the image needs to be loaded from the server and this causes a flash.



So what we do is use a single image **and shift the background position.**



And this is how it gets styled:



***What about a more complicated example:***

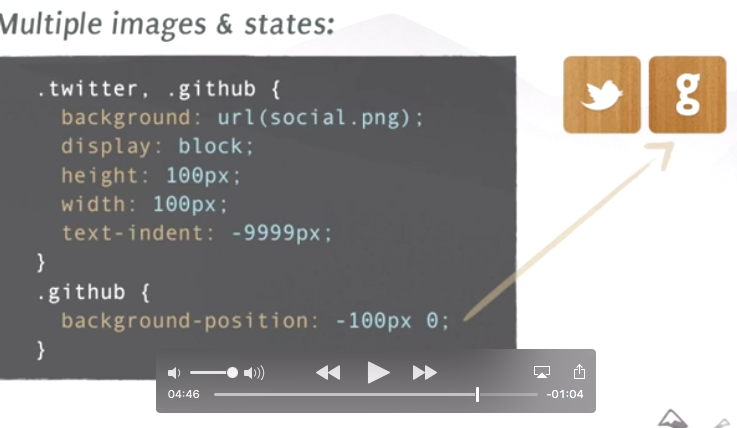


Each square is 100 px.

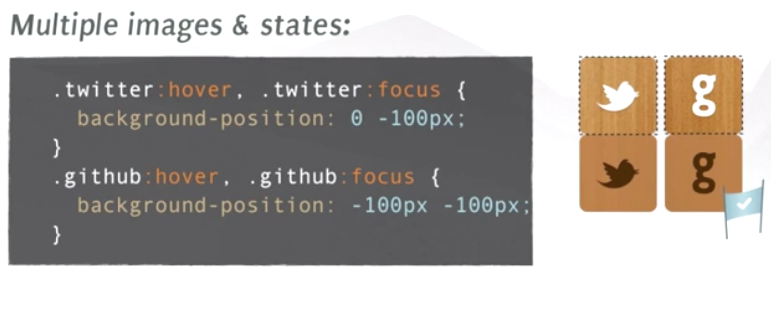


The initial styles are set up as described above. The height and width are set to 100 px and as the <a> element is defined twice. This means that two images are displayed.

We then do this. And this takes care of the resting state:



Now we add two focus states:



The following is a reccomended tool for dealing with Sprites:

<http://www.spritecow.com/>

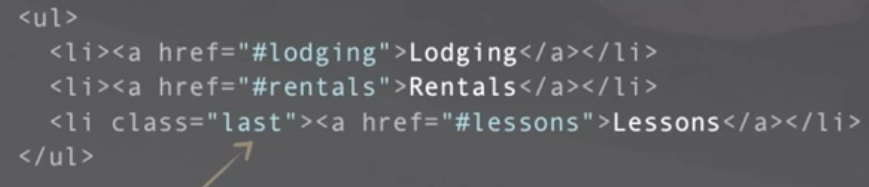
You can also use base 64 to encode the text.

Level 8 – Pseudo

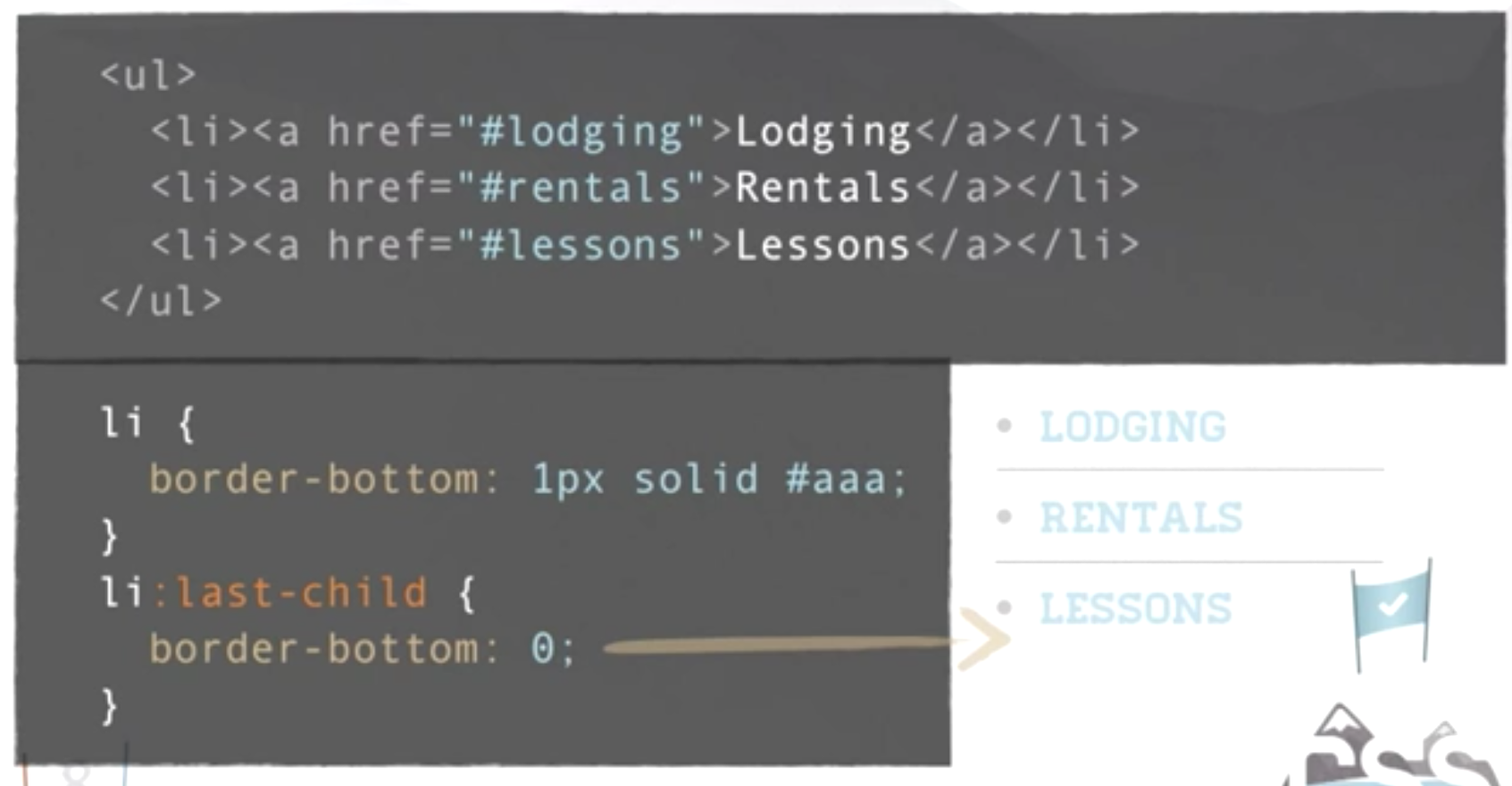
***Pseudo Classes***

Here is a problem, we have a list of items and then with the last item, we want the border to be nothing.

In the following a different class is used with the last <li> element in order to style it with a border bottom equal to nothing.



But a neater solution might be the following:



Another example is to have every second list item being a different color. We can do this manually. Or we can do it with the following:



There are other possibilities with nth-child. The following gives the basic formula:

<li:nth-child(an + b)

This will match items in intervals **A** starting with the element at position **B** (or 0 if b is not set. For example the following is even and then odd respectively:

<li:nth-child(2n) All Even elements (starts at 0)

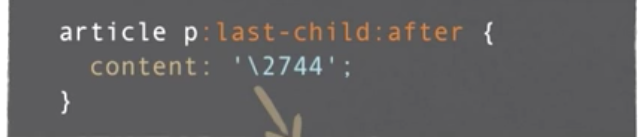
<li:nth-child(2n + 1) All Odd elements (starts at 1)

<li:nth-child(3n + 1) Every third element.

***Pseudo Elements***

There are also pseudo elements such as :first-child / :last-child / :only-child

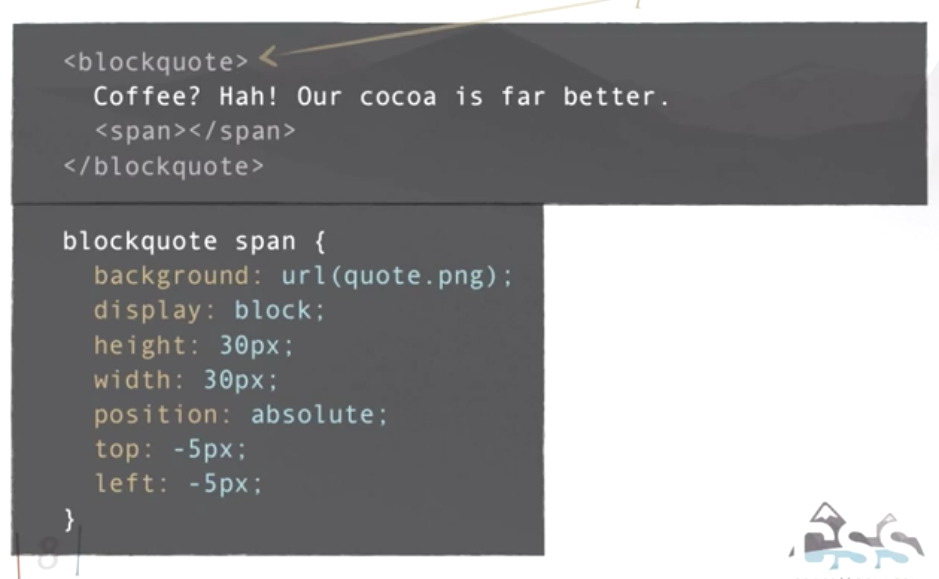
If we want the last paragraph to end with a decorative element…we could manually insert the unicode in the paragraph element. But we could do this as follows with CSS:



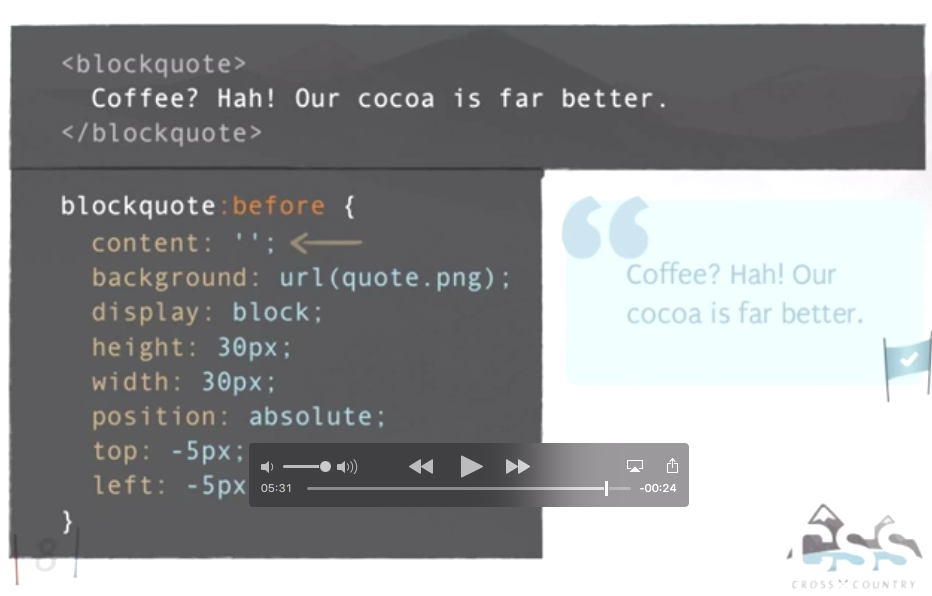
The above inserts a snowflake character.

Another example is the following:

We start with this:



Now we can do this (notice the reference to the image).



***Pseudo Element Links:***

<https://css-tricks.com/pseudo-element-roundup/>

<https://css-tricks.com/pseudo-class-selectors/>

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

***Supplemental Links:***

<https://css-tricks.com/pseudo-class-selectors/>

*Support for pseudo elements*

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

*Amazing stuff pseudo elements can do*

<https://css-tricks.com/pseudo-element-roundup/>