For applying style to  multiple element

:is(.card,h2) p:hover{

Color: red;

}

* 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

* **Content-box** : Default. The width and height properties (and min/max properties) includes only the content. Border and padding are not included
* **Border-box** : The width and height properties (and min/max properties) includes content, padding and border
* **Inline block in used by default in images**
* Inline : Displays an element as an inline element. Any height and width properties will have no effect.
* inline-block It’s formatted just like the inline element, where it doesn’t start on a new line. BUT, you can set width and height values.
* Pseudo-Classes: **Classes that are applied to elements based on information that is not present in the markup,** e.g. **:first-child or :last-child.** Do note that the selectors are **parsed from right to left** (see the demo). You cannot use section article:first-child to select the first occurrence of article, if the first child of section is h1 and not article. Likewise with the :nth-child, and :last-child pseudo-classes.
* FCP(First Contentful paint) measures how long it takes the browser to render the first piece of DOM content after a user navigates to your page. Images, non-white <canvas> elements, and SVGs on your page are considered DOM content; anything inside an iframe *isn't* included.
* FMP(First Meaningful Paint)  : When a website’s contents are fully loaded that users are looking for, that is called First Meaningful Paint.  This is the time at which the user feels that the primary content of the page is visible. As the name suggests after the first meaningful paint render, it giver meaningful information to the users.

a:link {

  color: #FF0000;

}

/\* visited link \*/

a:visited {

  color: #00FF00;

}

/\* mouse over link \*/

a:hover {

  color: #FF00FF;

}

a:active{

//test

}

* Pseudo-Elements: Pseudo-elements differ from Pseudo-Classes in that they **actually create an element in the document tree**. This is almost the first instance of CSS modifying the HTML document tree. You should ideally use **pseudo-elements with “::” instead of “:”** (but most browsers accept “:” notation for CSS 2.1 pseudo-elements). Pseudo-elements are: ::first-line, ::first-letter, ::before, ::after (See the demo for how pseudo-elements work).

**Defer and aysch in script tag:**

In other words:

* Scripts with **defer** never block the page.
* Scripts with **defer** always execute **when the DOM is read**y (but before **DOMContentLoaded** event).

* In other words, **async** scripts load in the background and run when ready.
* The DOM and other scripts don’t wait for them, and they don’t wait for anything. A fully independent script that runs when loaded. As simple, at it can get.

HOW to Optimze CSS Code?

* Use CDN
* Use image as font or svgs
* Minification to remove white space
* Load at the top

Combinator :

**What are Combinators?**

The selection of an element based on its occurrence in relation to another element (chosen by the choice of combinator: whitespace, >, +, or ~). You can have:

* Descendant Combinator
  + This is the most common usage, e.g. #lovelyweather h1.
* Child Combinator
  + Select an element if it is a direct child of another element (and not a grandchild of that element).
* Adjacent Sibling Combinator
  + The element that is immediately adjacent to another element.
* General Sibling Combinator
  + The element that is adjacent, but not immediately to another element.

**Explain the difference between visibility: hidden; and display: none;? What are the pros and cons of using display:none?**

* visibility: hidden simply hides the element but it will occupy space and **affect the layout of the document**.
* display: none removes the element from the normal layout flow (causes DOM reflow). It will not affect the layout of the document nor occupy space.

**How does z-index relate to positioning? Describe z-index and how stacking context is formed?**

* The z-index property specifies the stack order of elements. An element with a higher z-index stack order is always rendered in front of an element with a lower z-index stack order on the screen. z-index only works on positioned elements position: absolute, position: relative, or position: fixed. The default stack order of non-positioned elements is their order in the document.

**What is the purpose of the z-index and how is it used?**

* The z-index helps specify the stack order of positioned elements that may overlap one another. The z-index **default value is zero**, and can take on either a positive or negative number.
* An element with a higher z-index is always stacked above than a lower index.
* z-index can take the following values:
  + Auto: Sets the stack order equal to its parents.
  + Number: Orders the stack order.
  + Initial: Sets this property to its default value (0).
  + Inherit: Inherits this property from its parent element.

**How works absolute / relative / fixed / static position?**

**absolute**, place an element exactly where you want to place it. absolute position is actually **set relative to the element's parent.** if no parent available then relatively place to the page itself (it will default all the way back up to the element).

**relative**, means "**relative to itself"**. Setting position: relative; on an element and no other positioning attributes, it will no effect on it's positioning. It allows the use of z-index on the element and it limits the scope of absolutely positioned child elements. Any child element will be absolutely positioned within that block.

**fixed**, element is positioned relative to viewport or the browser window itself. viewport doesn't changed if u scroll and hence fixed element will stay right in the same position.

**static** default for every single page element. The only reason you would ever set an element to position: static is to **forcefully-remove some positioning that got applied to an element outside of your control**.**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:

**sticky** - Sticky positioning is a hybrid of relative and fixed positioning. The element is treated as relative positioned until it crosses a specified threshold, at which point it is treated as fixed positioned.

div.sticky {

  position: -webkit-sticky; /\* Safari \*/

  position: sticky;

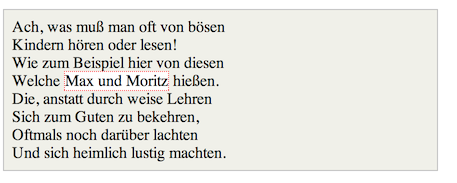
  top: 0;

}

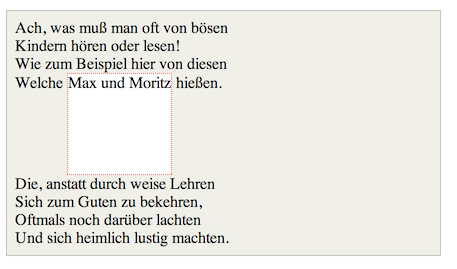
You must also specify at least one of top, right, bottom or left for sticky positioning to work.

Imagine a <span> element inside a <div>. If you give the <span> element a height of 100px and a red border for example, it will look like this with

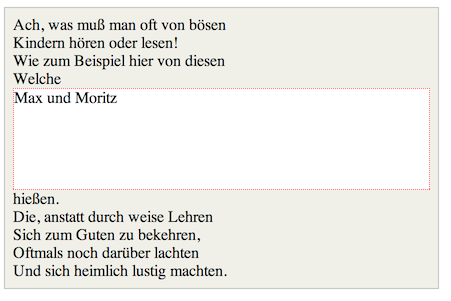
**display: inlinel**

****

**display: inline-block**

****

**display: block**

****

Code: <http://jsfiddle.net/Mta2b/>

Elements with display:inline-block are like display:inline elements, but they can have a **width** and a **height**. That means that you can use an inline-block element as a block while flowing it within text or other elements.

Difference of supported styles as summary:

* **inline: only margin-left, margin-right, padding-left, padding-right**
* **inline-block**: margin, padding, height, width

**Which one would you prefer among px, em % or pt and why?**

it depends on what you are trying to do.

* px gives fine grained control and maintains alignment because 1 px or multiple of 1 px is guaranteed to look sharp. px is not cascade, this means if parent font-size is 20px and child 16px. child would be 16px.
* em maintains relative size. you can have responsive fonts. em is the width of the letter 'm' in the selected typeface. However, this concept is tricky. 1em is equal to the current font-size of the element or the **browser default**. if u sent font-size to **16px** then **1em = 16px**. The common practice is to set default body font-size to 62.5% (equal to 10px). em is cascade
* % sets font-size relative to the font ol**size of the body.** Hence, you have to set font-size of the body to a reasonable size. this is easy to use and does cascade. for example, if parent font-size is 20px and child font-size is 50%. child would be 10px.
* pt(points) are traditionally used in print. 1pt = 1/72 inch and it is fixed-size unit

**Explain the CSS “box model” and the layout components that it consists of. Provide some usage examples.**

The CSS box model is a rectangular layout paradigm for HTML elements that consists of the following:  
Content The content of the box, where text and images appear  
Padding A transparent area surrounding the content (i.e., the amount of space between the border and the content)  
Border A border surrounding the padding (if any) and content  
Margin A transparent area surrounding the border (i.e., the amount of space between the border and any neighboring elements)  
/\* top   right  bottom left  \*/

padding: 25px  50px   75px   100px;

/\* same padding on all 4 sides: \*/

padding: 25px;

/\* top/bottom padding 25px; right/left padding 50px \*/

padding: 25px 50px;

/\* top padding 25px; right/left padding 50px; bottom padding 75px \*/

* padding: 25px 50px 75px;

**Explain what elements will match each of the following CSS selectors:**

* div, p Selects all <div> elements and all  
  elements
* div p Selects all <p> elements that are anywhere inside a  
  element
* div > p Selects all <p> elements where th**e immediate parent** is a  
  element
* div + p Selects all <p> elements that are placed **immediately after a  
  element**
* div ~ p Selects all <p> elements that are anywhere preceded by a  
  element

**Explain the meaning of each of these CSS units for expressing length:**

cm centimeters em elements (i.e., relative to the font-size of the element; e.g., 2 em means 2 times the current font size) in inches mm millimeters pc picas (1 pc = 12 pt = 1/6th of an inch) pt points (1 pt = 1/72nd of an inch) px pixels (1 px = 1/96th of an inch)

**In CSS3, how would you select:**

* Every <a> element whose href attribute value begins with “https”.
  + a[href^="https"]
* Every <a> element whose href attribute value ends with “.pdf”.
  + a[href$=".pdf"]
* Every <a> element whose href attribute value contains the substring “css”.
  + a[href\*="css"]

With the CSS box-sizing Property

The box-sizing property allows us to **include the padding and border** in an element's total width and height.

**If you set box-sizing: border-box; on an element padding and border are included in the width and height:**

**What is CSS preprocessor and why to user one?**

* A CSS preprocessor is a program that lets you generate CSS from the preprocessor's own unique syntax. There are many CSS preprocessors to choose from, however most CSS preprocessors will add some features that don't exist in pure CSS, **such as mixin, nesting selector, inheritance selector,** and so on. These features make the CSS structure more readable and easier to maintain.
* Here are a few of the most popular CSS preprocessors:
  + SASS (SCSS)
* **What are the advantages/disadvantages of using CSS preprocessors?**
* *Advantages:*
* \* CSS is made more maintainable.
* \* Easy to write nested selectors.
* \* Variables for consistent theming. Can share theme files across different projects.
* \* Mixins to generate repeated CSS.
* \* Splitting your code into multiple files. CSS files can be split up too but doing so will require an HTTP request to download each CSS file.
* *Disadvantages:*
* \* Requires tools for preprocessing. Re-compilation time can be slow.
* + LESS
  + Stylus
  + PostCSS

**What are the some pseudo classed u have used?**

* pseudo class tells you specific state of an element. allow to style element dynamically. The most popular one is :hover. Besides i have used :visited, :focus, :nth-child, nth-of-type, :link, etc.
* pseudo classes is better if you don't want to mess up with javaScript however, pseudo-classes is slow to process and apply rules

**List as many values for the display property that you can remember.**

* none, block, inline, inline-block, table, table-row, table-cell, list-item.

**Have you played around with the new CSS Flexbox or Grid specs?**

* Flexbox or Flexible boxes, is a new layout mode in CSS3 Use of flexbox ensures that elements behave predictably when the page layout must accommodate **different screen sizes and different display devices**.
* For many applications, the flexible box model provides an improvement over the block model in that it does not use floats, nor do the flex container's margins collapse with the margins of its contents.
* Grid specs CSS Grid Layout is a specification for creating two-dimensional grids Grid is a companion to the Flexible Box Module (flexbox). Flexbox is designed for one-dimensional layout, so things can be arranged in an unbroken line. Grid is designed for two-dimensional layout, meaning the items don't need to sit next to each other. In the future we're likely to use both: **Grid Layout for main page areas, and flexbox for the smaller UI elements it excels with.**

**What is an iframe and how it works?**

* An iframe is an HTML document which can be embedded inside another HTML page.
* The IFrame HTML element is often used to insert content from another source, such as an advertisement, into a Web page.

<iframe src="https://github.com" height="300px" width="300px"></iframe>

**How is responsive design different from adaptive design?**

Both responsive and adaptive design attempt to optimize the user experience across different devices, adjusting for different viewport sizes, resolutions, usage contexts, control mechanisms, and so on.

Responsive design works on the principle of flexibility - a single fluid website that can look good on any device. Responsive websites use media queries, flexible grids, and responsive images to create a user experience that flexes and changes based on a multitude of factors. Like a single ball growing or shrinking to fit through several different hoops.

Adaptive design is more like the modern definition of progressive enhancement. Instead of one flexible design, adaptive design detects the device and other features and then provides the appropriate feature and layout based on a predefined set of viewport sizes and other characteristics. The site detects the type of device used and delivers the pre-set layout for that device. Instead of a single ball going through several different-sized hoops, you'd have several different balls to use depending on the hoop size.

**Which property is used to change the face of a font?**

* The **font-family** property is used to change the face of a font.

**Which property is used to make a font italic or oblique?**

* The **font-style** property is used to make a font italic or oblique.

**Which property is used to underline, overline, and strikethrough text?**

* The text-decoration property is used to underline, overline, and strikethrough text.

**Which property is used to capitalize text or convert text to uppercase or lowercase letters?**

* The text-transform property is used to capitalize text or convert text to uppercase or lowercase letters.

**What is specificity? How do u calculate specificity?**

* a process of determining which css rule will be applied to an element. it actually determines which rules will take precedence.
* inline style usually wins then ID then class value (or pseudo-class or attribute selector), universal selector (\*) has no specificity.

**What is the difference between em and rem units?**

Both em and rem units are based on the font-size CSS property. The only difference is where they inherit their values from.

* em units inherit their value from the font-size of the **parent element**
* rem units inherit their value from the font-size of the **root element (html)**

In most browsers, the font-size of the root element is set to 16px by default.

**What are data- attributes good for?**

* The HTML5 data attribute lets you assign custom data to an element. When we want to store more information/data about the element when no suitable HTML5 element or attribute exists

**Consider HTML5 as an open web platform. What are the building blocks of HTML5?**

* more semantic text markup
* new form elements
* new video and audio elements
* javascript API
* canvas and SVG
* geolocation API
* new data storage

**Describe the difference between <script>, <script async> and <script defer>.**

* Normal Execution: script: Parsing of the HTML code pauses while the script is executing. For slow servers and heavy scripts this means that displaying the webpage will be delayed.
* script defer Delays script execution until the HTML parser has finished. A positive effect of this attribute is that the DOM will be available for your script. However, since not every browser supports defer yet, can’t rely on it.
* Asynchronous Execution: script async HTML parsing may continue and the script will be executed as soon as it’s ready.

**SSR VS CSR**

* We are using server side rendering for two reasons:
  + **performance benefit for our customers**
  + **Consistent SEO performance**
* The main difference is that for SSR your server’s response to the browser is the HTML of your page that is ready to be rendered, while for CSR the browser gets a pretty empty document with links to your javascript. That means for SSR your browser will start rendering the HTML from your server without having to wait for all the JavaScript to be downloaded and executed.
* for SSR, the user can start viewing the page while all of that is happening. For the CSR world, you need to wait for all of the above to happen and then have the virtual dom moved to the browser dom for the page to be viewable.

**What is the difference between a <span> and a <div>?**

* <div> is a block level element which means it will render it on it's own line with a width of a **100% of the parent element.**
* <span> is an inline element which means it will render on the same line as the previous element, **if it is also an inline element, and it's width will be determined by it's content.**

**Name 5 common block-level and inline HTML elements.**

* block elements <h1>, <p>, <ul>, <ol>, <li>,
* inline elements **<span>, <a>, <strong>, <i>, <img>**

**What does CORS stand for and what issue does it address?**

Cross Origin Resource Sharing. To address the fact that browsers restrict cross-origin HTTP requests initiated from within scripts. CORS gives web servers cross-domain access controls, which enable secure cross-domain data transfers.

**How does the browser rendering engine work?**

In order to render content the browser has to go through a series of steps:

* Document Object Model(DOM)
* CSS object model(CSSOM)
* Render Tree
* Layout
* Paint.

Decreasing Load Time :

* Optimizing cache
* Apply Ajax
* Trim down javascript parsing
* Do away with redirect
* Minify javascript
* Using CDN
* Using social button

Aligning content in center :

.container {

  position: relative;

}

.child {

**position: absolute;**

**top: 50%;**

**left: 50%;**

**transform: translate(-50%, -50%);**

}

//ysing flexbox :

.container {

  ...

  display: flex;

  justify-content: center;

  align-items: center;

}

New HTML5 APIs (Application Programming Interfaces)

The most interesting new APIs in HTML5 are:

* HTML Geolocation
* HTML Drag and Drop
* HTML Local Storage
* HTML Application Cache
* HTML Web Workers
* HTML SSE : -

var source = new EventSource("demo\_sse.php");

source.onmessage = function(event) {

  document.getElementById("result").innerHTML += event.data + "<br>";

};

**HTML5Shiv**

Thankfully, Sjoerd Visscher created the HTML5Shiv! The HTML5Shiv is a JavaScript workaround to enable styling of HTML5 elements in versions of Internet Explorer prior to version 9.

<head>

  <!--[if lt IE 9]>

<script src="/js/html5shiv.js"></script>

  <![endif]-->

</head>

**CANVAS**

The <canvas> tag is used to draw graphics, on the fly, via scripting (usually JavaScript).

The <canvas> tag is only a container for graphics, you must use a script to actually draw the graphics.

<canvas id="myCanvas"></canvas>

<script>

var canvas = document.getElementById("myCanvas");

var ctx = canvas.getContext("2d");

ctx.fillStyle = "#FF0000";

ctx.fillRect(0, 0, 80, 80);

</script>

Differences Between SVG and Canvas

**SVG is a language for describing 2D graphics in XML.**

Canvas draws 2D graphics, on the fly (with a JavaScript).

**SVG is XML based, which means that every element is available within the SVG DOM. You can attach JavaScript event handlers for an element.**

In SVG, each drawn shape is remembered as an object. If attributes of an SVG object are changed, the browser can automatically re-render the shape.

**Canvas is rendered pixel by pixel.** In canvas, once the graphic is drawn, it is forgotten by the browser. If its position should be changed, the entire scene needs to be redrawn, including any objects that might have been covered by the graphic.

Comparison of Canvas and SVG

The table below shows some important differences between Canvas and SVG:

|  |  |
| --- | --- |
| **Canvas** | **SVG** |
| * Resolution dependent * **No support for event handlers** * Poor text rendering capabilities * You can save the resulting image as .png or .jpg * Well suited for graphic-intensive games | * Resolution independent * **Support for event handlers** * Best suited for applications with large rendering areas (Google Maps) * Slow rendering if complex (anything that uses the DOM a lot will be slow) * Not suited for game applications |

What are 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.

<meta name="viewport" content="width=device-width, initial-scale=1.0">

**WEBSTORAGE**

Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

The sessionStorage object is equal to the localStorage object, except that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.

The following example counts the number of times a user has clicked a button, in the current session:

**How Browser Loads page and execute Scripts ?**

Here's what happens when a browser loads a website with a <script> tag on it:

1. Fetch the HTML page (e.g. index.html)
2. Begin parsing the HTML
3. The parser encounters a <script> tag referencing an external script file.
4. The browser requests the script file. Meanwhile, the parser blocks and stops parsing the other HTML on your page.
5. After some time the script is downloaded and subsequently executed.
6. The parser continues parsing the rest of the HTML document.

Step #4 causes a bad user experience. Your website basically stops loading until you've downloaded all scripts. If there's one thing that users hate it's waiting for a website to load.

Why does this even happen?

Any script can insert its own HTML via document.write() or other DOM manipulations. This implies that the parser has to wait until the script has been downloaded & executed before it can safely parse the rest of the document. After all, the script *could* have inserted its own HTML in the document.

However, most JavaScript developers no longer manipulate the DOM *while* the document is loading. Instead, they wait until the document has been loaded before modifying it. For example:

<!-- index.html --> <html> <head> <title>My Page</title> <script type="text/javascript" src="my-script.js"></script> </head> <body> <div id="user-greeting">Welcome back, user</div> </body> </html>

Javascript:

// my-script.js document.addEventListener("DOMContentLoaded", function() { // this function runs when the DOM is ready, i.e. when the document has been parsed document.getElementById("user-greeting").textContent = "Welcome back, Bart"; });

Because your browser does not know my-script.js isn't going to modify the document until it has been downloaded & executed, the parser stops parsing.

Antiquated recommendation

The old approach to solving this problem was to put <script> tags at the bottom of your <body>, because this ensures the parser isn't blocked until the very end.

This approach has its own problem: the browser cannot start downloading the scripts until the entire document is parsed. For larger websites with large scripts & stylesheets, being able to download the script as soon as possible is very important for performance. If your website doesn't load within 2 seconds, people will go to another website.

In an optimal solution, the browser would start downloading your scripts as soon as possible, while at the same time parsing the rest of your document.

The modern approach

Today, browsers support the async and defer attributes on scripts. These attributes tell the browser it's safe to continue parsing while the scripts are being downloaded.

CSS :

Mix-blend-mode : multiply or screen for cutout effect

This property specifies how much of the remaining space in the flex container should be assigned to the item (the flex grow factor).

FLEXBOX :

.container {

display: flex;

flex-direction: row | row-reverse | column | column-reverse;

flex-grow: 4;

flex-wrap: nowrap | wrap | wrap-reverse;

justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly | start | end | left | right ... + safe | unsafe;

align-items: stretch | flex-start | flex-end | center | baseline | first baseline | last baseline | start | end | self-start | self-end + ... safe | unsafe;

}

GRID :

.container {

  display: grid | inline-grid;

grid-template-columns: 40px 50px auto 50px 40px;

  grid-template-rows: 25% 100px auto;

}

. What are the new form elements introduced in HTML5?

Ans. The new form elements introduced in HTML5 are:

* <datalist> – specifies a list of options for input controls
* <keygen> – generates an encryption key
* <output> – defines the result of an expression
* <progress> – heads only in the direction of 100% of the max value
* <meter> – provides for a gauge, displaying a general value within a range

 What is an ETag?

 What is Cross-site Scripting (XSS)? How to prevent it?

What is long polling?

Ans. Long polling is a technique of having a persistent connection with the server. In long polling, the client polls the server requesting new information. The server holds a client’s connection open for as long as possible. The connection is closed only after the data is sent back to the client or connection timeout occurs.

Long Polling Flow:

* A request is sent to the server.
* The server holds a client’s connection until new data is available.
* The server responds to the request when the new information appears.
* The browser immediately sends another request.