**What is Position Absolute?**

It works when there is parent child relationship between the tags.

Position absolute element always looks for the immediate coming nonstatic postion element.

When it finds a nonstatic parent it can move around inside that parent body.

If there is no nonstatic parent element the absolute element takes the web pages as its parent and move around it.

Absolute-positioned elements are completely taken out of the regular flow of the web page.

They are not positioned based on their usual place in the document flow, but based on the position of their ancestor.

In the example above, the absolutely positioned square is inside a statically positioned parent.

This means it will be positioned relative to the whole page itself, which means relative to the <html> element – the root of the page.

The coordinates, top: 50px; and left: 0;, are therefore based on the whole page.

If you want the coordinates to be applied to its parent element, you need to relatively position the parent element by updating .parent while keeping .one the same:

.parent {

width: 500px;

border: 1px solid red;

margin: auto;

text-align: center;

position: relative;

}

.one {

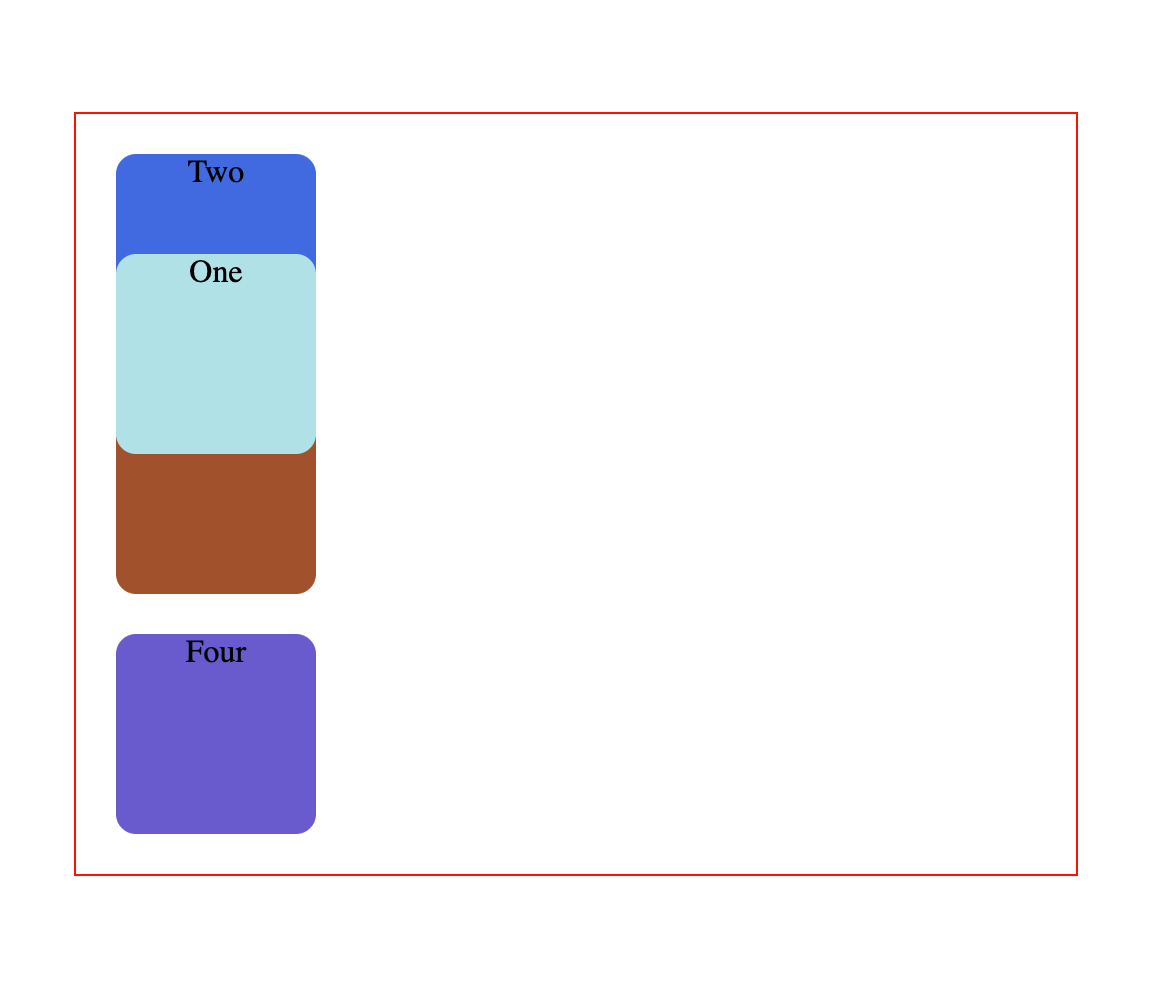
background-color: powderblue;

position: absolute;

top: 50px;

left: 0;

}

This code creates the below result

Absolute positioning takes elements out of the regular document flow while also affecting the layout of the other elements on the page.

It seems like you're discussing the behavior of absolutely positioned elements in CSS, particularly in relation to their parent elements. Here's a clearer explanation:

Absolute Positioning in CSS

Absolute Positioning:

When an element is positioned with position: absolute;, it is removed from the normal document flow and positioned relative to its nearest positioned ancestor (an ancestor with a position value other than static).

If there is no such ancestor, the element is positioned relative to the initial containing block (usually the browser window).

Key Points:

100ms Delay: To animate the position change, you might use a CSS transition with a delay of 100ms. For example\

.absolute-element {

position: absolute;

transition: top 100ms, left 100ms;

}

Non-static Parent: If the absolutely positioned element finds a parent with position: relative;, position: absolute;, position: fixed;, or position: sticky;, it will position itself relative to that parent. If all ancestors are position: static;, it will use the viewport as its reference.

Example:

<!DOCTYPE html>

<html lang="en">

[**1. In how many ways can we add CSS to our HTML file?**](https://www.geeksforgeeks.org/types-of-css-cascading-style-sheet)

Cascading Style Sheet(CSS) is used to set the style in web pages that contain HTML elements. It sets the background color, font size, font family, color, … etc properties of elements on a web page.  
There are three types of CSS which are given below:

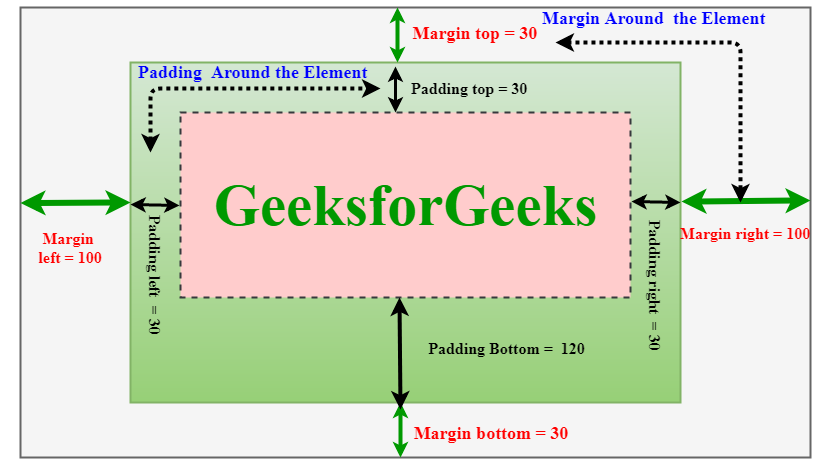
* **Inline CSS:** Inline CSS contains the CSS property in the body section attached with the element known as inline CSS. This kind of style is specified within an HTML tag using the style attribute.
* **Internal or Embedded CSS:** This can be used when a single HTML document must be styled uniquely. The CSS ruleset should be within the HTML file in the head section i.e the CSS is embedded within the HTML file.
* **External CSS:** External CSS contains a separate CSS file which contains only style property with the help of tag attributes (For example class, id, heading, … etc). CSS property is written in a separate file with .css extension and should be linked to the HTML document using the **link** tag. This means that for each element, style can be set only once and that will be applied across web pages.

[**2. Which type of CSS holds the highest priority?**](https://www.geeksforgeeks.org/types-of-css-cascading-style-sheet)

Inline CSS has the highest priority, then comes Internal/Embedded followed by External CSS which has the least priority. Multiple style sheets can be defined on one page. If for an HTML tag, styles are defined in multiple style sheets then the below order will be followed.

* As Inline has the highest priority, any styles that are defined in the internal and external style sheets are overridden by Inline styles.
* Internal or Embedded stands second in the priority list and overrides the styles in the external style sheet.
* External style sheets have the least priority. If there are no styles defined either in the inline or internal style sheet then external style sheet rules are applied for the HTML tags.

[**3. What is the difference between margin and padding?**](https://www.geeksforgeeks.org/css-margins-padding)

* Margin is used to create space around elements and padding is used to create space around elements inside the border.
* We can set the margin property to auto but we cannot set the padding property to auto.
* In Margin property we can allow negative or float number but in padding we cannot allow negative values.
* Margin and padding target all the 4 sides of the element. Margin and padding will work without the border property also. The difference will be more clear with the following example.

[**4. What is CSS Box Model?**](https://www.geeksforgeeks.org/css-box-model)

The CSS box model is a container that contains multiple properties including borders, margin, padding, and the content itself. It is used to create the design and layout of web pages. It can be used as a toolkit for customizing the layout of different elements. The web browser renders every element as a rectangular box according to the CSS box model.  
Box-Model has multiple properties in CSS. Some of them are given below:

* borders
* margins
* padding
* Content

The following figure illustrates the box model.



* **Border Area:** It is the area between the box’s padding and margin. Its dimensions are given by the width and height of the border.
* **Margin Area:** This area consists of space between border and margin. The dimensions of the Margin area are the margin-box width and the margin-box height. It is useful to separate the element from its neighbors.
* **Padding Area:** It includes the element’s padding. This area is actually the space around the content area and within the border box. Its dimensions are given by the width of the padding-box and the height of the padding-box.
* **Content Area:** This area consists of content like text, images, or other media content. It is bounded by the content edge and its dimensions are given by content box width and height.

**5. What are pseudo-classes in CSS?**

A Pseudo class in CSS is used to define the special state of an element. It can be combined with a CSS selector to add an effect to existing elements based on their states. For Example, changing the style of an element when the user hovers over it, or when a link is visited. All of these can be done using Pseudo Classes in CSS.

**Note**that pseudo-class names are not case-sensitive.

**Syntax:**

selector: pseudo-class{  
 property: value;  
}

There are many Pseudo-classes in CSS but the ones which are most commonly used are as follows:

* **:hover Pseudo-class:** This pseudo-class is used to add a special effect to an element when our mouse pointer is over it. The below example demonstrates that when your mouse enters the box area, its background color changes from yellow to orange.
* **:active Pseudo-class:** This pseudo-class is used to select an element that is activated when the user clicks on it. The following example demonstrates that when you click on the box, its background color changes for a moment.
* **:focus Pseudo-class:** This pseudo-class is used to select an element that is currently focussed by the user. It works on user input elements used in forms and is triggered as soon as the user clicks on it. In the following example, the background color of the input field which is currently focused changes.
* **:visited Pseudo-class:** This pseudo-class is used to select the links which have been already visited by the user. In the following example, the color of the link changes once it is visited.

**6. What are pseudo-elements in CSS?**

**Pseudo Elements:** Pseudo-element in CSS is used to add style to specified parts of an element. Example: Using style before or after an element.

**Syntax:**

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

**Use of Pseudo-Element:** Below is some examples to describe the use of pseudo-element.

* **::before Pseudo-element:**It is used to add some CSS property before an element when that element is called.
* **::after Pseudo-element:**It is used to add some CSS property after an element when that element is called.
* **::first-letter Pseudo-element:**It is used to make changes to the first letter of an element.
* **::first-line Pseudo-element:**It is used to make changes to the first line of an element.

**7) What is Flex Layout Model ?**

The Flexbox layout model in CSS is designed to help you create flexible and responsive layouts with ease. It provides a more efficient way to layout, align, and distribute space among items in a container, even when their sizes are unknown or dynamic. Here's an overview of the Flexbox layout model and its core concepts:

**Flex Container:**

* The parent element of the flex items. To create a flex container, apply display: flex; or display: inline-flex; to the parent element.

 **flex-direction:**

* row: Aligns flex items horizontally (default).
* column: Aligns flex items vertically.
* row-reverse: Aligns flex items horizontally in reverse order.
* column-reverse: Aligns flex items vertically in reverse order.

8) Flex Box Model ?

The [flexible box layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout) module (usually referred to as flexbox) is a one-dimensional layout model for distributing space between items.

9) Flex Grow, Flex Shrink, Flex Basis ?

**flex-grow:** This property defines how much a flex item will grow relative to the other flex items inside the flex container. It takes a unitless value, which acts as a proportion. For example, if you have flex-grow: 1 on one item and flex-grow: 2 on another, the second item will grow twice as much as the first one.

.item {

flex-grow: 1;

}

**flex-shrink**: This property defines how much a flex item will shrink relative to the other flex items inside the flex container. It also takes a unitless value, which acts as a proportion. For example, if you have flex-shrink: 1 on one item and flex-shrink: 2 on another, the second item will shrink twice as much as the first one when there is not enough space.

css

Copy code

.item {

flex-shrink: 1;

}

**Type of Css Units :**

CSS units have two basic types:

* Absolute units
* Relative units

Absolute units are fixed and do not depend on the size of the parent element or the viewport. Examples of absolute units are pixels (px), points (pt), and centimeters (cm).

Relative units, on the other hand, are flexible – and just as the name implies, they are relative to the parent element’s size, the viewport’s size, or the root element’s font size.

Since there are a number of CSS unit types, you might have trouble deciding which unit to use for a particular measurement. This article will demonstrate the best use cases for each of these units. We'll focus on the most important and frequently used CSS units here: rem, em, vh, vw, %, and the all too familiar absolute unit – px.

**Rem and em?**

**Rem (rem):**

The rem unit in CSS stands for "root em". It is a relative unit of measurement that is relative to the font size of the root element. One rem is equal to the font size of the root element. The root element defaults to 16px in many browsers, so 1rem is equal to 16px.

<div class="container">

<h1>Best Practices for CSS units</h1>

<p>This is a paragraph with font size set to 2rem</p>

</div>

html {

font-size: 16px;

}

.container {

margin: 20px;

padding: 20px;

border: 1px solid #ddd;

}

h1 {

font-size: 2rem;

color: #0077cc;

}

p {

font-size: 1rem;

color: #0077cc;

}

The h1 element is set to 2rem which means that it's two times the root element, the html element. 2 x 16px = 32, so the h1 element is 32px.

Here's the root element at 16px:

**Em:**

Similar to rem, em is a relative unit of measurement. But unlike rem, em is relative to the font size of the parent element or the font-size of the nearest parent with a defined font size.

Let's look at another example:

<div class="parent-element">

This is the nearest parent

<div class="child-element">

<p>This is the child</p>

</div>

</div>

.parent-element {

font-size: 20px;

margin: 20px;

padding: 20px;

border: 1px solid #ddd;

}

p {

font-size: 2em;

color: #0077cc;

}

Here, the font size of the child-element is set to 2 times that of the parent element. That makes it 40px (2 x 20).

**Percentages (%):**

Percentages are relative units that render an element relative to the size of the element's parent. They serve as a percentage of their containing block and are always relative to their nearest parent.

**Viewport height (vh):**

vh is a relative unit of measurement that represents the visible height area of the browser window.

**Px:**

px is useful when you want to specify a fixed size of an element, such as a border size or an image size.

**GRID :**

The grid-template-columns and grid-template-rows properties are fundamental in CSS Grid Layout.

**1. fr (Fractional Unit)**

**When to Use:**

* **Proportional Layouts**: Use fr when you want columns or rows to take up a proportion of available space. This is useful when designing flexible layouts that adjust to varying screen sizes.
* **Dynamic Content**: When you need columns or rows to automatically adjust their size based on the remaining space in the grid container.

**Example:**

.grid-container {

display: grid;

grid-template-columns: 1fr 2fr 1fr; /\* Three columns with proportional widths \*/

}

In this example:

* The second column takes twice as much space as the first and third columns.

**2. auto**

**When to Use:**

* **Content-Based Sizing**: Use auto when you want the column or row size to be based on the content it holds. This ensures that the size adjusts to fit the content without exceeding a specific width or height.
* **Dynamic Content Size**: When content size is unpredictable, and you want the grid item to resize accordingly.

**Example:**

.grid-container {

display: grid;

grid-template-columns: auto auto auto; /\* Three columns sized based on content \*/

}

In this example:

* Each column will be as wide as its content requires, up to the maximum available space.

**3. Fixed Units (px, em, %)**

**When to Use:**

* **Precise Control**: Use fixed units when you need precise control over the size of columns or rows, regardless of the content.
* **Stable Layout**: When you want columns or rows to have a specific size that doesn’t change with the viewport or content size.

**Example:**

.grid-container {

display: grid;

grid-template-columns: 100px 200px 150px; /\* Three columns with fixed widths \*/

}

In this example:

* The first column is 100px wide, the second is 200px, and the third is 150px.

**Choosing the Right Approach**

* **Responsive Design**: For responsive designs, use fr units to create flexible layouts that adjust based on the available space. Combine with auto or fixed units as needed.
* **Content-Driven Layouts**: Use auto if your columns or rows need to fit around content without exceeding available space.
* **Fixed Layouts**: Use fixed units if you need precise control over the size of your grid items and don’t need them to adapt dynamically.

**Margin-left and left Difference?**

**Left**:

It doesnot afftect surrounding layout.

Only used by positioned elements

Position the element in the container

It does not accepts the auto as value.

**Margin-left:**

It afftect surrounding layout.

used by all elements

add space around the components.

It accepts the auto as value.

Css precedence?

**z-index work on relative position elements?**

The z-index property only takes effect on elements that are positioned. This means that for z-index to work, the element must have a position value of either:

* relative
* absolute
* fixed
* sticky

By default, all elements have a position value of static. Static elements are not affected by the z-index property. Therefore, if an element is positioned as static, the z-index will not apply to it, and it will be stacked according to the order in which it appears in the HTML.

23) Combinators:

**1. Descendant Combinator ( )**

**CSS Rule**: div p { color: red; }

**HTML Example**:

<div>

<p>This paragraph will be red (descendant).</p>

<div>

<p>This paragraph will also be red (nested descendant).</p>

</div>

</div>

**Explanation**:

* The rule targets all <p> elements that are descendants of a <div>, regardless of how deeply nested they are. Therefore, both paragraphs in the example will be red.

**2. Child Combinator (>)**

**CSS Rule**: div > p { color: blue; }

**HTML Example**:

<div>

<p>This paragraph will be blue (direct child).</p>

<span>

<p>This paragraph will NOT be blue (not a direct child).</p>

</span>

</div>

**Explanation**:

* The rule applies only to <p> elements that are direct children of a <div>. In this case, the first paragraph will be blue, while the second paragraph (inside the <span>) will not be affected.

**3. Adjacent Sibling Combinator (+)**

**CSS Rule**: h1 + p { margin-top: 0; }

**HTML Example**:

<h1>This is a heading</h1>

<p>This paragraph will have no top margin (adjacent sibling).</p>

<p>This paragraph will have a default margin.</p>

**Explanation**:

* The rule targets the first <p> element immediately following an <h1>. This paragraph will have its top margin set to 0, while the second paragraph will retain its default margin.

**4. General Sibling Combinator (~)**

**CSS Rule**: h1 ~ p { color: green; }

**HTML Example**:

<h1>This is a heading</h1>

<p>This paragraph will be green (general sibling).</p>

<p>This paragraph will also be green (general sibling).</p>

**Explanation**:

* The rule applies to all <p> elements that are siblings following an <h1>. Therefore, both paragraphs after the <h1> will be styled green.

Object Fit :

The object-fit property is used to specify how an <img> or <video> should be resized to fit its container.

This property tells the content to fill the container in a variety of ways; such as "preserve that aspect ratio" or "stretch up and take up as much space as possible".