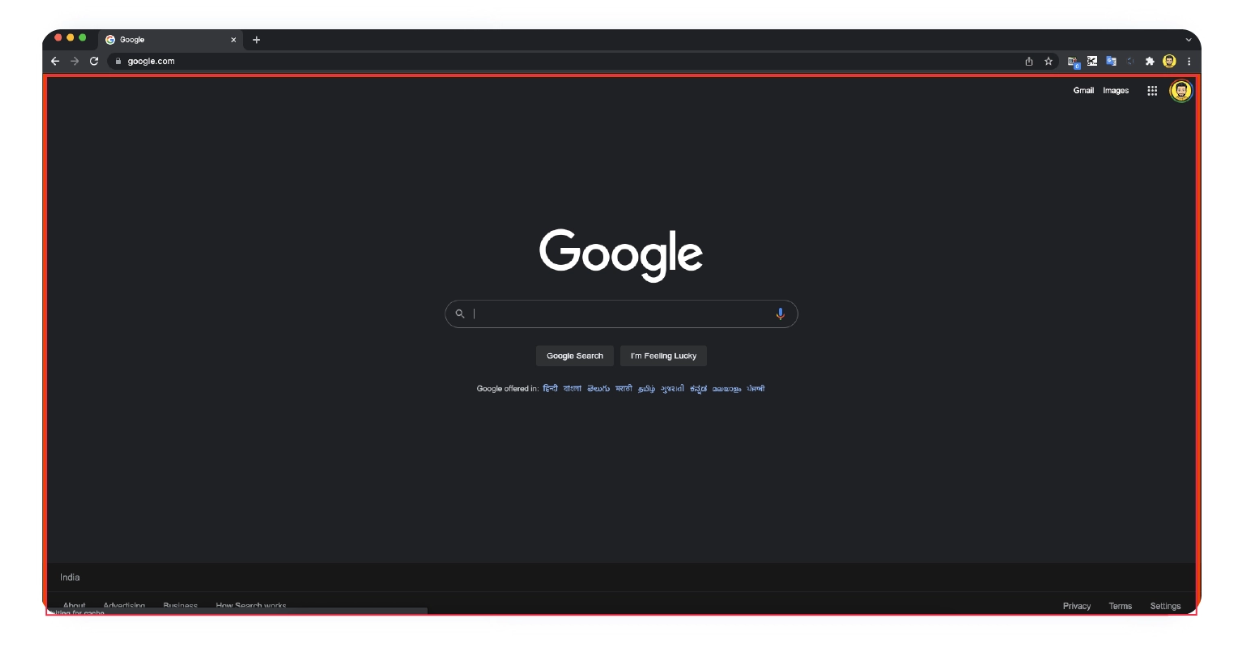
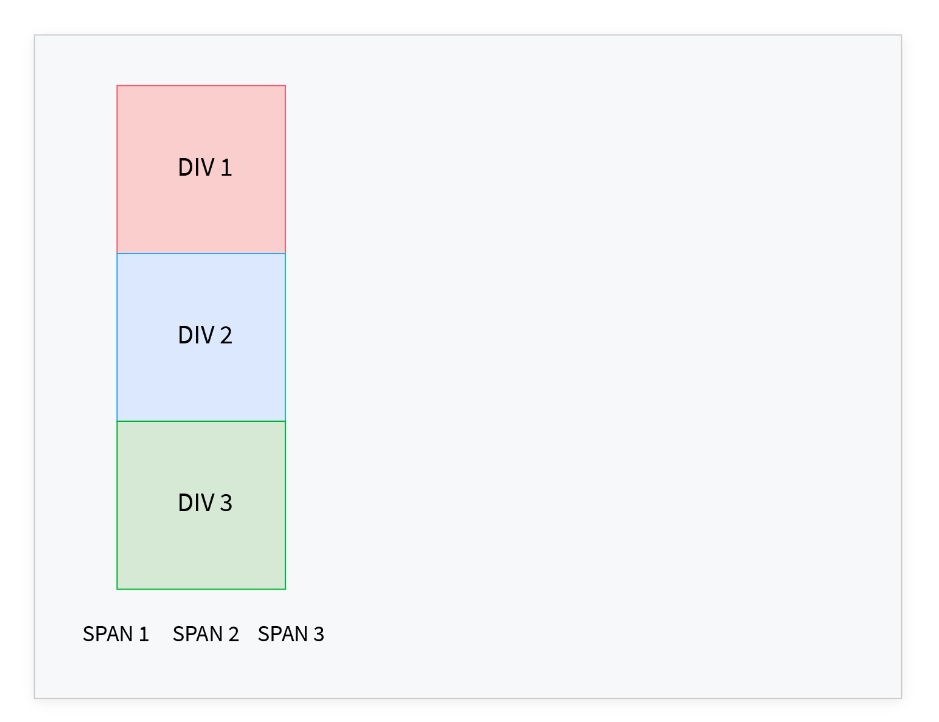
**CSS Position Property:**

The CSS position property defines the position of an HTML element inside the browser's viewport. By default, all the HTML elements will follow a default order. We can use the CSS position property to manipulate the position of HTML elements inside the viewport.

All the HTML elements take a specific position in a browser's viewport. **Viewport is the area of the browser window that contains the web content**. (The area inside the rectangle in the below image).



By default, the browser renders the HTML elements one after the other in the viewport. The ordering can be horizontal or vertical, depending on the element type. For example, all div elements are rendered vertically, and all span elements are rendered horizontally.



**The CSS position property defines the position of an HTML element inside the viewport**. The position property is used with left, right, top, bottom, and z-index properties to position the elements in the viewport.

## Property Values

The CSS position property accepts five values used to determine the position of an HTML element.

* static
* relative
* absolute
* fixed
* sticky

### Static:

All HTML elements are positioned static by default. With static positioning, the elements are positioned along with the natural flow of the document. The properties left, right, top, bottom, and z-index do not affect the elements positioned as static.

**Syntax**

selector {

position: static;

}

**Example**

Let's create three divs with the colors red, blue, and green.

**HTML**

<html>

<head>

<title>CSS Positions</title>

</head>

<body>

<div class="red"></div>

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

<div class="green"></div

</body>

</html>

**CSS**

We set the position property to static and the left property to 100px for the **blue** div.

:root {

--red: #E74C36;

--blue: #2980B9;

--green: #27AE60;

}

div {

width: 100px;

height: 100px;

}

.red {

background-color: var(--red);

}

.blue {

background-color: var(--blue);

position: static;

left: 100px;

}

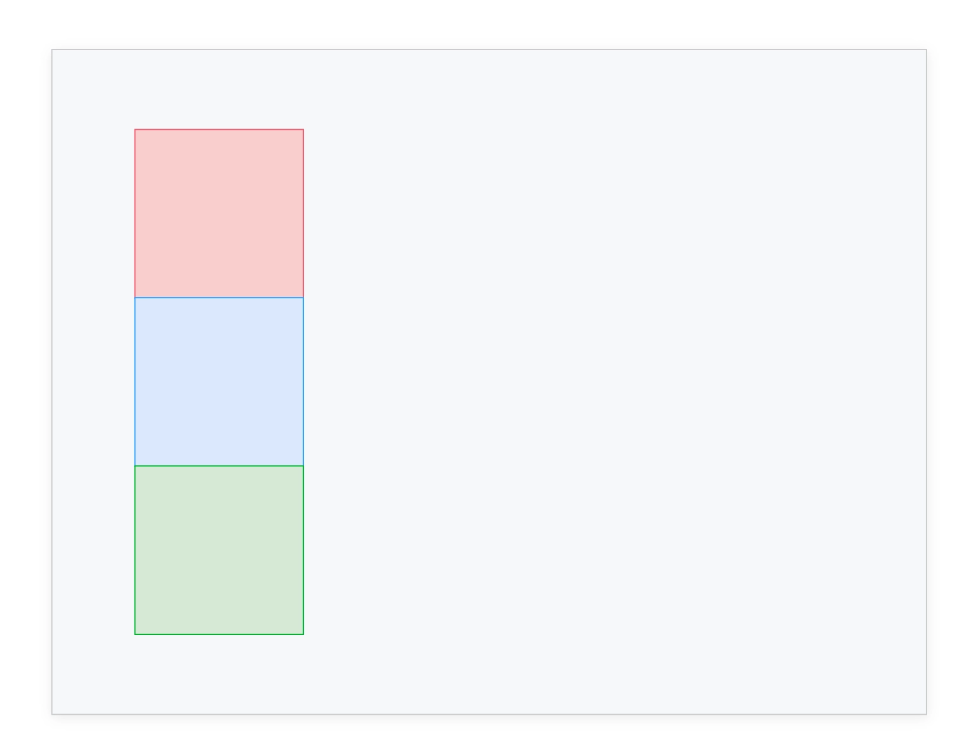
.green {

background-color: var(--green);

}

**Output:**

Eventhough the left property is set to 100px for the blue div, it doesn't alter its position in the viewport.



**Relative:**

With relative positioning, the elements are positioned along with the natural flow of the document. But unlike static elements, the position of the relative elements can be modified using left, right, top, bottom, and z-index properties.

**Syntax:**

selector {

position: relative;

}

**Example:**

Now for the same HTML content we saw above, we are setting the position property to relative and the left property to 100px for the blue div

**CSS:**

.blue {

background-color: var(--blue);

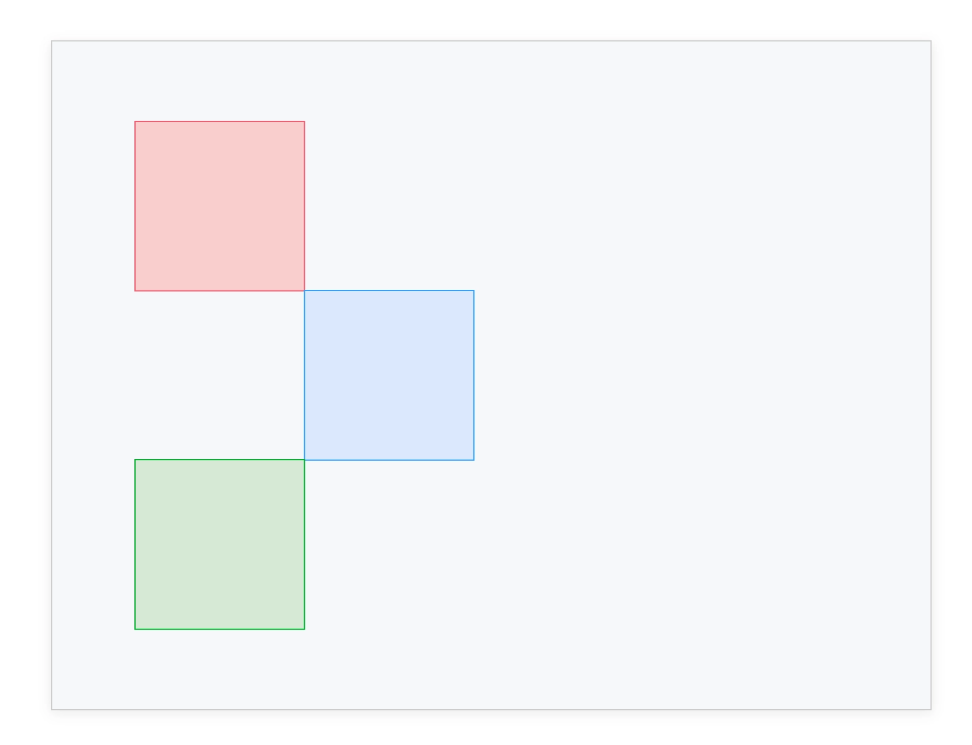
position: relative;

left: 100px;

}

**Output:**

We can see that the blue div is moved 200px from the left. Even though it is moved from the left, there is a space between the red and the green div. This is because **the relative positioning doesn't remove the element from its default position**. Instead, the element is moved to the given position and its default place left unoccupied.



**Absolute:**

The elements are positioned relative to their parent elements with absolute positioning. **The absolute elements are positioned relative to the closest ancestor with any position property other than static**. If the closest ancestor has a static position, the element is positioned relative to the next parent element without the static position property.

**Example:**

Let's set position: relative to the **blue div** and give the value of left to 100px.

**CSS:**

.blue {

background-color: var(--blue);

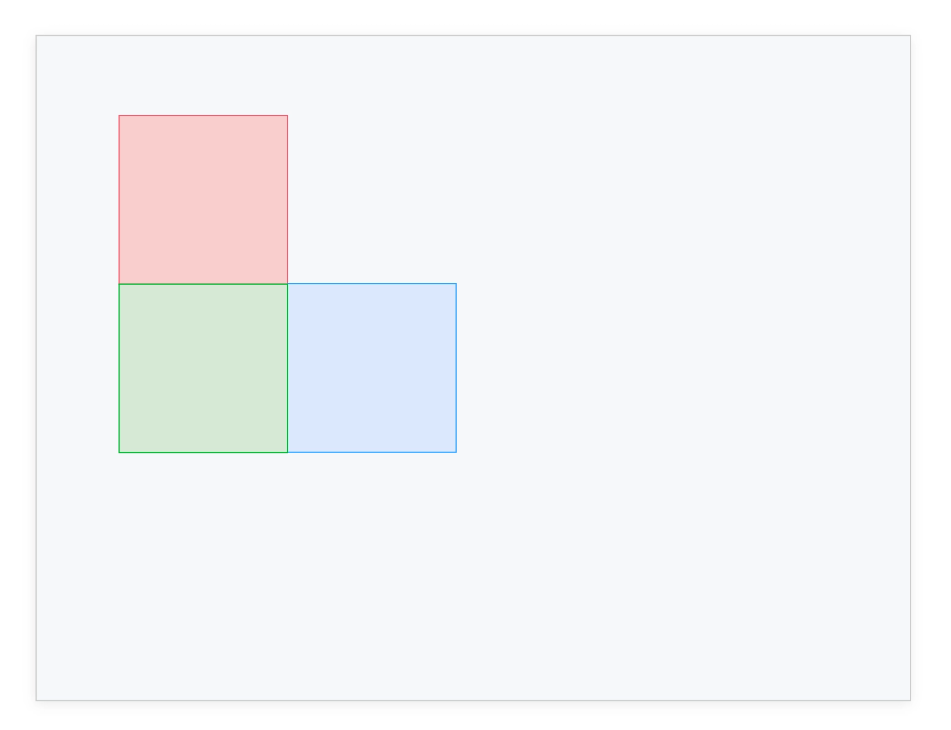
position: absolute;

left: 100px;

}

**Output:**

We can see that the **blue div** is moved 200px from the left. But unlike relative elements, **the absolute element is removed from its default position and moved to the new position**. Hence its old position is occupied by the next element. In this case, it is the green div.



**Fixed:**

Elements with fixed positioning are always positioned relative to the HTML element (i.e.) the root of the document. The fixed elements stay in the same position irrespective of the scrolling. Like absolute elements, the fixed elements are also removed from the natural flow of the document, and other elements occupy their place.

**Example:**

In this example, we set position: fixed and top: 200px for the blue div.

**CSS:**

.blue {

background-color: var(--blue);

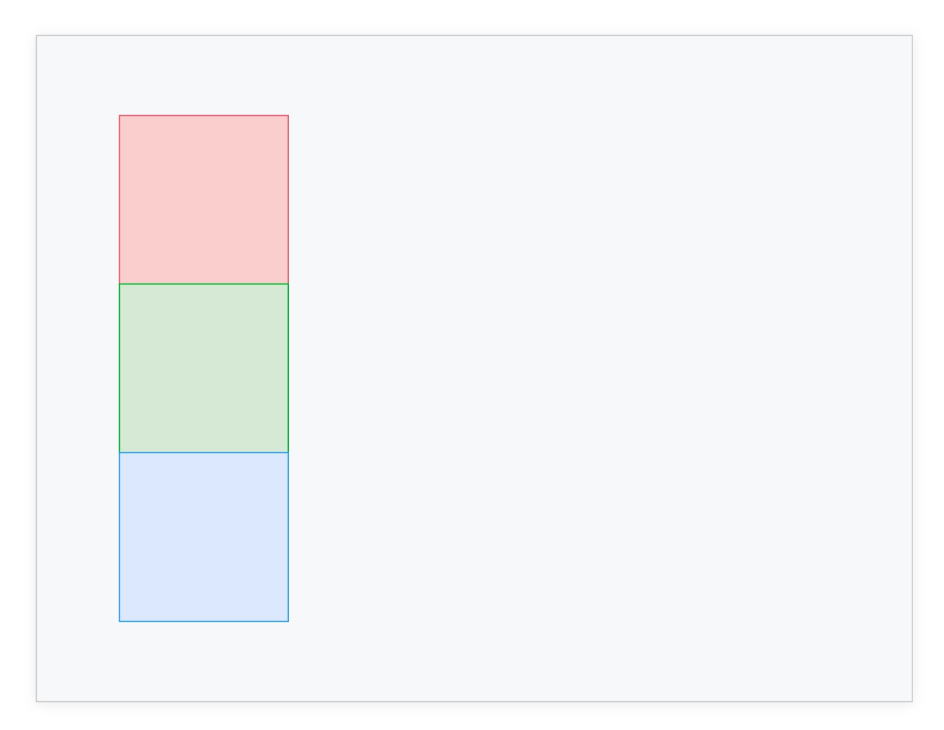
position: fixed;

top: 200px;

}

**Output:**

We can see that the blue div is positioned 200px from the top and the green div takes the place of the blue div. So the order changed from **red => blue => green** to **red => green => blue**.



**Sticky:**

With sticky positioning, the element behaves like a relative positioned until a certain scroll point, and then it will be fixed.

**Example:**

In this example, we set height: 1000px for body, so that we will have scroll bar and position: sticky, top: 50px for the blue div.

**CSS:**

body {

height: 1000px;

}

.blue {

background-color: var(--blue);

position: sticky;

top: 50px;

}

**Output:**

When we scroll, the blue div scrolls to some extent, and then it is fixed, and then the green div continues to scroll.

**Understanding the z-index Property**

* The z-index property defines the order of the overlapping HTML elements. Elements with a **higher z-index value will be placed on top of the elements with a lower z-index value**. The z-index property only works on the positioned elements (relative, absolute, fixed, and static). z-index doesn't affect the elements with position: static.

**How to Use the z-index Property?**

* The z-index value can be set with an integer value. Elements with a higher value will be placed on top of those with a lower value.

**Example:**

We set position: absolute and top: 0px for all the divs. This stack contains all the three divs on top of one another. We set different left values to make everything visible. We applied the z-index value to 3, 2, and 1 for the red, blue, and green divs, respectively. This makes **the three divs stacked in the order red => blue and blue => green, red on top, and green on the bottom**.

**CSS:**

div {

width: 100px;

height: 100px;

position: absolute;

top: 0px;

}

.red {

background-color: var(--red);

left: 10px;

z-index: 3;

}

.blue {

background-color: var(--blue);

left: 20px;

z-index: 2;

}

.green {

background-color: var(--green);

left: 30px;

z-index: 1;

}