AIMERZ.ai
Aim.Act.Achieve

NOTES

Position in CSS

Introduction to CSS Layout

In CSS, layout is all about how the elements on a webpage are arranged. CSS provides various ways to control the structure and positioning of HTML elements. By understanding how CSS layout works, you can build responsive, flexible, and visually appealing web pages.

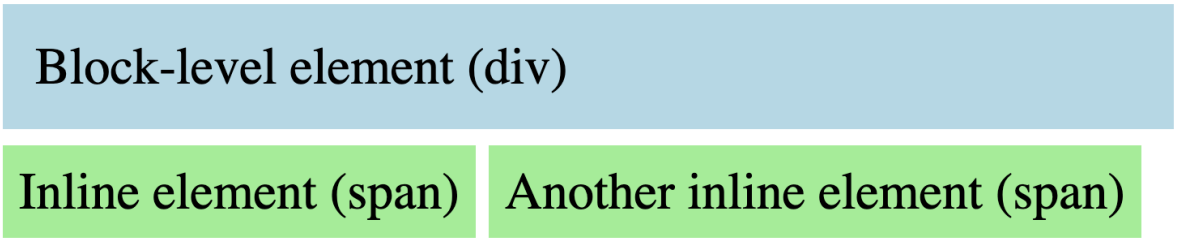
Normal Flow

The **normal flow** of a webpage refers to how elements are displayed by default. HTML elements are typically displayed from top to bottom, and within each line, they are displayed **left to right. Block-level elements (like , <p>, <h1>)** take up the full width available, while **inline** elements **(like , <a>)** only take up as much space as they need.

Let us take an example :

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Normal Flow Example</title>
    <style>
        div {
            background-color: lightblue;
            padding: 10px;
            margin-bottom: 10px;
        }
        span {
            background-color: lightgreen;
            padding: 5px;
        }
    </style>
</head>
<body>
    <div>Block-level element (div)</div>
    <span>Inline element (span)</span>
    <span>Another inline element (span)</span>
</body>
</html>
```

Output:



In the above example , the is a block-level element and takes up the full width.

The **** elements are inline, and they only take up as much space as they need, appearing on the same line unless the width runs out.

Display & Visibility Properties

Display Property

The display property is used to define how an element is displayed in the layout. Below are some common display values:

1. **Block:** The element takes up the full width and starts on a new line.

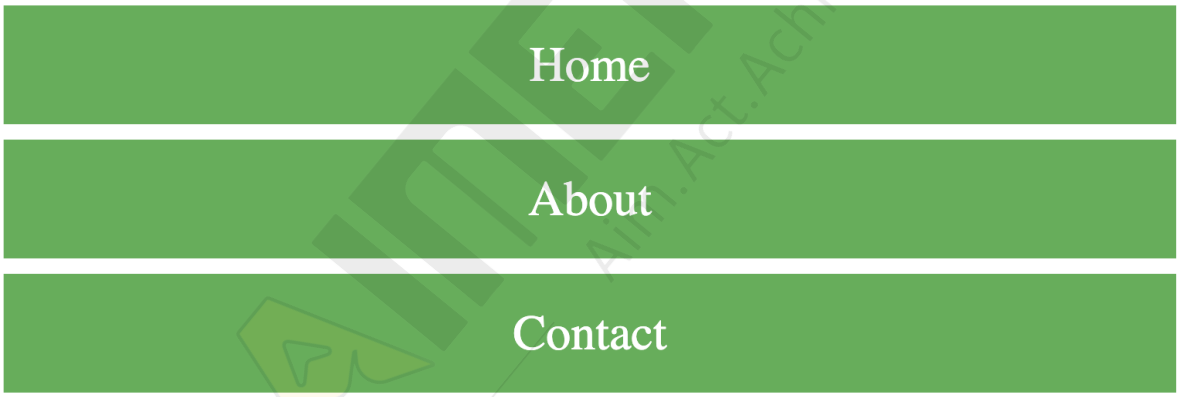
Example:

In a navigation menu, we often use block elements for links.

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Navigation Menu</title>
    <style>
        .nav-link {
```

```
            display: block;
            background-color: #4CAF50;
            color: white;
            text-align: center;
            padding: 10px;
            margin: 5px 0;
            text-decoration: none;
        }
    </style>
</head>
<body>
    <a href="#" class="nav-link">Home</a>
    <a href="#" class="nav-link">About</a>
    <a href="#" class="nav-link">Contact</a>
</body>
</html>
```

Output:



In the above example, each link takes up the full width of its container due to **display: block.** The links stack vertically and are styled as block-level elements.

2. **inline**: The element takes up only as much width as its content, and it doesn't start on a new line.

Example: In a footer, links are often displayed inline.

```
JavaScript
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Footer Links</title>
    <style>
        .footer-link {
            display: inline;
            color: #333;
            text-decoration: none;
            margin-right: 15px;
        }
    </style>
</head>
<body>
    <footer>
        <a href="#" class="footer-link">Privacy Policy</a>
        <a href="#" class="footer-link">Terms of Service</a>
        <a href="#" class="footer-link">Help</a>
    </footer>
</body>
</html>
```

Output:

3. **inline-block**: The element behaves like an inline element but can have width and height applied, unlike regular inline elements.

Ex:

```
JavaScript
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Product Cards</title>
    <style>
      .product {
        display: inline-block;
        width: 200px;
        margin: 10px;
        padding: 10px;
        border: 1px solid #ccc;
        text-align: center;
      }
      .product img {
        width: 100%;
        height: auto;
      }
      .product h3 {
        font-size: 18px;
        margin: 10px 0;
      }
      .product p {
        color: grey;
      }
    </style>
  </head>
  <body>
    <div class="product">
      <img

src="https://images.pexels.com/photos/8532616/pexels-photo-8532616.jpeg?auto
=compress&cs=tinysrgb&w=800"
        alt="Product 1"
      />
      <h3>Product 1</h3>
      <p>$20.00</p>
    </div>
    <div class="product">
      <img

src="https://media.istockphoto.com/id/2152217447/photo/a-person-is-looking-t
hrough-a-rack-of-clothes-concept-of-election.jpg?b=1&s=612x612&w=0&k=20&c=ps
bZFZAfpNPlPeE6_2L1GeBw448wn9PCSBVezSxZxQU="
        alt="Product 2"
      />
      <h3>Product 2</h3>
      <p>$25.00</p>
    </div>
```

```
    <div class="product">
      <img

src="https://media.istockphoto.com/id/1048701936/photo/young-man-standing-co
nfidently.jpg?b=1&s=612x612&w=0&k=20&c=YrQ6ePWnnGwdmE7jTOVyUAp3BEcWq-zVQvQkn
cMcVdI="
        alt="Product 3"
      />
      <h3>Product 3</h3>
      <p>$30.00</p>
    </div>
  </body>
</html>
```

4. **Table**: The **display: table** property makes an element behave like a table, enabling table-like layout for elements without using the traditional <table> element. This is particularly useful for designing structured content, like pricing tables or comparison charts

Ex:

JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Pricing Table Example</title>
    <style>
      .pricing-table {
        display: table;
        width: 100%;
        background-color: lightgray;
      }
      .pricing-row {
        display: table-row;
      }
      .pricing-cell {
        display: table-cell;
        padding: 15px;
```

```
        border: 1px solid black;
      }
    </style>
  </head>
  <body>
    <div class="pricing-table">
      <div class="pricing-row">
        <div class="pricing-cell">Plan</div>
        <div class="pricing-cell">Price</div>
        <div class="pricing-cell">Features</div>
      </div>
      <div class="pricing-row">
        <div class="pricing-cell">Basic</div>
        <div class="pricing-cell">$10/month</div>
        <div class="pricing-cell">Feature A, Feature B</div>
      </div>
      <div class="pricing-row">
        <div class="pricing-cell">Premium</div>
        <div class="pricing-cell">$30/month</div>
        <div class="pricing-cell">All Features</div>
      </div>
    </div>
  </body>
</html>
```

Output:

Plan	Price	Features
Basic	$10/month	Feature A, Feature B
Premium	$30/month	All Features

5. **Display: None :** The display: none property removes an element from the document layout entirely. It doesn't take up any space and behaves as though it doesn't exist on the page.

Let's take an real-life example to understand better:

Hide Newsletter Signup After Submission

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Hide Newsletter</title>

  </head>
  <body>
    <div class="newsletter">
      <h2>Sign up for our Newsletter</h2>
      <input type="email" placeholder="Enter your email" />
      <button>Subscribe</button>
    </div>
    <p>
      The newsletter section is hidden because the user has already
subscribed.
    </p>
  </body>
</html>
```

Output:

Now use display:none in the above example:

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Hide Newsletter</title>
    <style>
      .newsletter {
        display: none;
      }
    </style>
  </head>
  <body>
    <div class="newsletter">
      <h2>Sign up for our Newsletter</h2>
      <input type="email" placeholder="Enter your email" />
      <button>Subscribe</button>
    </div>
    <p>
      The newsletter section is hidden because the user has already
subscribed.
    </p>
  </body>
</html>
```

Output:

The newsletter section is hidden because the user has already subscribed.

Visibility Property

The visibility property allows an element to be hidden or shown, but unlike display: none, hidden elements with visibility still take up space in the layout.

1. **Visibility: visible**

   By default, elements have visibility: visible, meaning they are shown and occupy space as expected.

```javascript
.element {
    visibility: visible;
}
```

This is the default behavior, so no special CSS is needed unless you are overriding a previous state.

2. **Visibility:hidden**

   This makes the element invisible, but it still occupies space on the page.

Example: Temporarily Hide a Button

```javascript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
    <title>Visibility Hidden Example</title>
    <style>
      .hidden-button {
        visibility: hidden;
      }
    </style>
  </head>
  <body>
    <button class="hidden-button">Submit</button>
    <p>The button is hidden but still takes up space on the page.</p>
  </body>
</html>
```

Output:

The button is hidden but still takes up space on the page.

In the above example, the button is hidden using **visibility: hidde**n, but the space it would have taken remains on the page. This might be useful for temporarily disabling or hiding elements without shifting the layout.

3. Visibility: collapse

This property is mainly used for table rows or columns. When a row or column is collapsed, it disappears without leaving any space behind.

Example: Collapsing a Table Row

JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Table Row Collapse</title>
    <style>
        table, th, td {
            border: 1px solid black;
            padding: 10px;
        }
        .collapsed-row {
            visibility: collapse;
        }
    </style>
</head>
<body>
    <table>
        <tr>
            <th>Item</th>
            <th>Price</th>
        </tr>
        <tr>
            <td>Item 1</td>
            <td>$10</td>
        </tr>
        <tr class="collapsed-row">
            <td>Item 2</td>
            <td>$20</td>
        </tr>
        <tr>
            <td>Item 3</td>
            <td>$30</td>
        </tr>
    </table>
</body>
</html>
```

Output:

Item	Price
Item 1	$10
Item 3	$30

In the above, the row with Item 2 is collapsed using visibility: collapse. It doesn't take up any space in the table, effectively removing the row from view while keeping the other rows intact.

Display: None vs. Visibility: Hidden vs. Opacity: 0

These three CSS techniques are often confused, but they serve different purposes.

A) Display: None

It completely removes the element from the page.

Example: Use when you don't want the element to appear at all.

B) Visibility: Hidden

It hides the element but leaves the space it occupies intact.

Example: Use when you want to hide something temporarily but maintain layout consistency.

C) Opacity: 0

- it makes the element fully transparent (invisible) but still interactive and taking up space.

Example: Use when you want the element to be invisible but still able to interact with, or for animations.

Let's take an example to understand better

Imagine you're on a blog or e-commerce site. The site offers a discount or updates via newsletter, and you want to sign up. Once you enter your email and submit, the form disappears, and a thank-you message takes its place.

index.html

JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Newsletter Signup Example</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container newsletter-submitted">
      <!-- Newsletter Form -->
      <div class="newsletter">
        <h2>Sign Up for Our Newsletter</h2>
        <p>Get the latest updates and offers directly to your inbox.</p>
        <form action="#">
          <input
            type="email"
            id="email"
            name="email"
            placeholder="Enter your email"
            required
          />
          <br />
          <button type="submit">Subscribe</button>
        </form>
      </div>

      <!-- Success message after submission -->
      <div class="success-message">
        <h2>Thank You!</h2>
        <p>You have successfully subscribed to our newsletter.</p>
      </div>
    </div>
  </body>
</html>
```

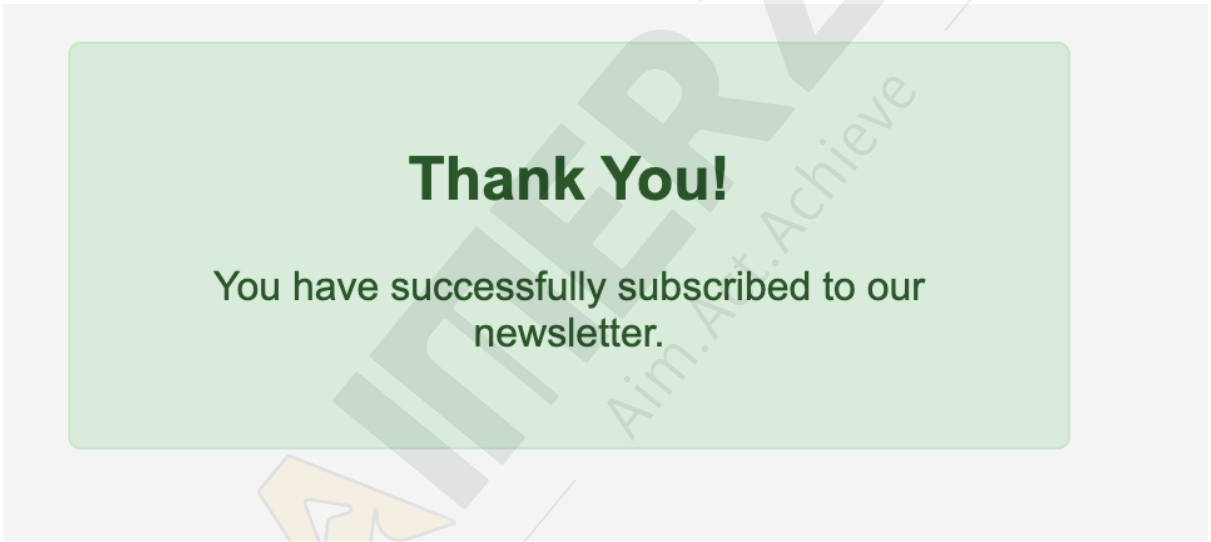
style.css

JavaScript

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  padding: 20px;
}

.container {
  width: 400px;
  margin: 0 auto;
}

.newsletter {
  background-color: #fff;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 5px;
  text-align: center;
  margin-bottom: 20px;
}

.newsletter input[type="email"] {
  width: 80%;
  padding: 10px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.newsletter button {
  padding: 10px 20px;
  background-color: #28a745;
  color: #fff;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

/* Form is hidden after submission */
.newsletter-submitted .newsletter {
  display: none;
}

.success-message {
  text-align: center;
  background-color: #d4edda;
```

```
  color: #155724;
  padding: 20px;
  border: 1px solid #c3e6cb;
  border-radius: 5px;
  display: none; /* Hidden by default */
}

/* Success message shown after submission */
.newsletter-submitted .success-message {
  display: block;
}
```

Output:



Float

The float property is primarily used for wrapping text around images or creating layouts where elements align to the left or right of their container.

Float Property Values

1. **left**: Floats the element to the left.
2. **right**: Floats the element to the right.
3. **none**: The element does not float (default value).
4. **inherit**: Inherits the float value from its parent.

We set the float layout using the float property.

```JavaScript
float: left;
float:right;
```

Ex: Float the div on the right of the paragraph , as shown below

index.html

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Float Property</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <p>
      <div class="float-div"></div>
      Lorem, ipsum dolor sit amet consectetur adipisicing elit. Perferendis
      tempore alias quod voluptatibus, ducimus aliquam ut laborum nihil in
ab
      labore non inventore debitis distinctio omnis quibusdam temporibus
```

```
      reprehenderit aliquid sapiente fugiat accusamus maxime sunt? Tempore
      officiis reiciendis nam numquam asperiores architecto, corporis
aliquid
      accusantium, exercitationem odio iusto soluta id.
    </p>
  </body>
</html>
```

style.css

```
JavaScript
.float-div {
  float: right;
  width: 150px;
  height: 150px;
  margin-left: 15px;
  background-color: chocolate;
}
```

Output:

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Perferendis tempore alias quod voluptatibus, ducimus aliquam ut laborum nihil in ab labore non inventore debitis distinctio omnis quibusdam temporibus reprehenderit aliquid sapiente fugiat accusamus maxime sunt? Tempore officiis reiciendis nam numquam asperiores architecto, corporis aliquid accusantium, exercitationem odio iusto soluta id.

Clear Property

By using the clear CSS property, you can determine whether an element needs to be cleared from the floating elements that come before it.

It can take the following values:

- **right:** The element is placed below the right-floated element.
- **left:** The element is placed below the left-floated element.
- **both:** The element is placed below both left and right-floated elements.
- **none:** The element is not moved down to clear the space.
- **inherit:** The element inherits the clear value from its parent.

Let us take an example to understand better:

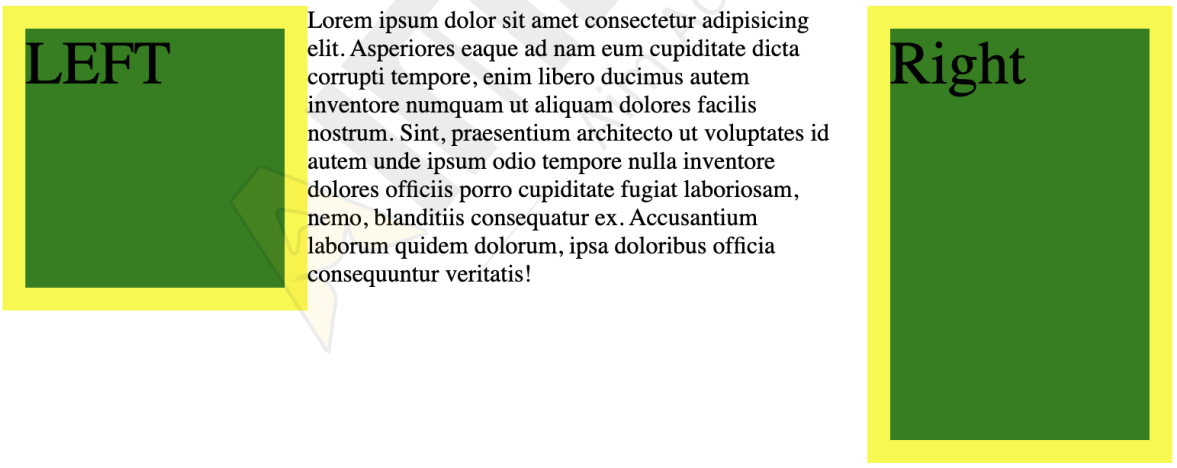
index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <div class="float-left">LEFT</div>
      <div class="float-right">Right</div>
      <p class="para">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Asperiores
        eaque ad nam eum cupiditate dicta corrupti tempore, enim libero
ducimus
        autem inventore numquam ut aliquam dolores facilis nostrum. Sint,
        praesentium architecto ut voluptates id autem unde ipsum odio
tempore
        nulla inventore dolores officiis porro cupiditate fugiat laboriosam,
        nemo, blanditiis consequatur ex. Accusantium laborum quidem dolorum,
        ipsa doloribus officia consequuntur veritatis!
      </p>
    </div>
  </body>
</html>
```

style.css

```
JavaScript
.float-left {
  float: left;
  width: 170px;
  height: 170px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}
.float-right {
  float: right;
  width: 170px;
  height: 270px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}
```

Output:



In the above HTML, We have to focus on three elements, float-left, float-right and para. Now let's see the difference in output by changing the clear property as left,right and both , none one by one.

Let's take some example to understand better

clear: none

index.html

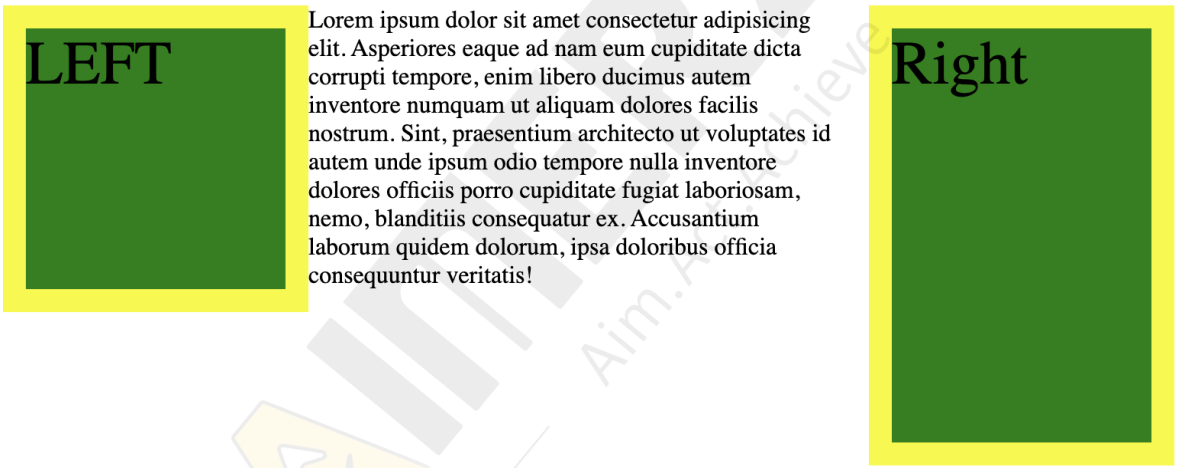
```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <div class="float-left">LEFT</div>
      <div class="float-right">Right</div>
      <p class="para">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Asperiores
        eaque ad nam eum cupiditate dicta corrupti tempore, enim libero
ducimus
        autem inventore numquam ut aliquam dolores facilis nostrum. Sint,
        praesentium architecto ut voluptates id autem unde ipsum odio
tempore
        nulla inventore dolores officiis porro cupiditate fugiat laboriosam,
        nemo, blanditiis consequatur ex. Accusantium laborum quidem dolorum,
        ipsa doloribus officia consequuntur veritatis!
      </p>
    </div>
  </body>
</html>
```

style.css

```
JavaScript
.float-left {
  float: left;
  width: 170px;
  height: 170px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}
```

```
.float-right {
  float: right;
  width: 170px;
  height: 270px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}

.para {
  clear: none;
}
```

Output:



clear:left

JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
```

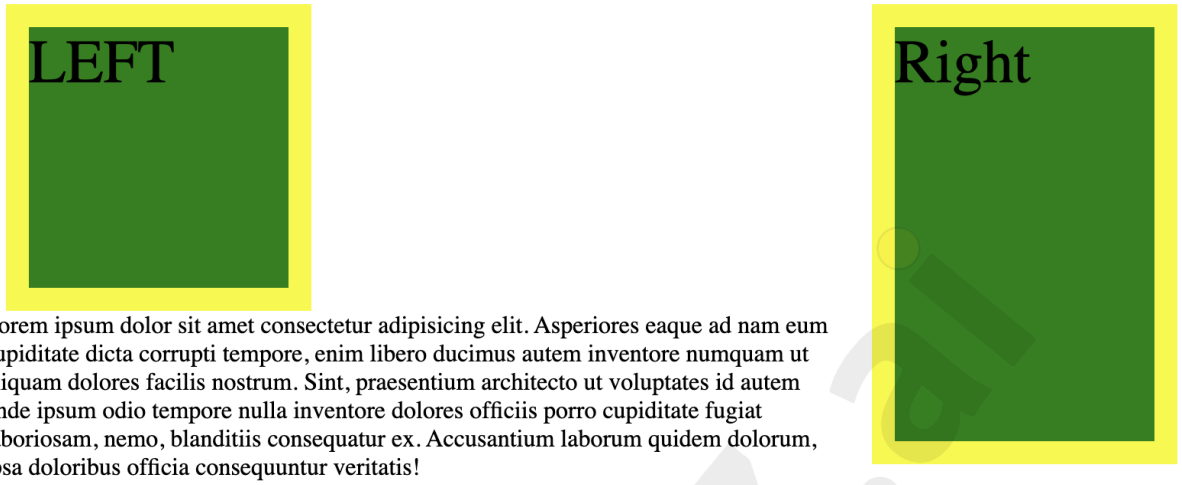
```
    <div>
      <div class="float-left">LEFT</div>
      <div class="float-right">Right</div>
      <p class="para">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Asperiores
        eaque ad nam eum cupiditate dicta corrupti tempore, enim libero
ducimus
        autem inventore numquam ut aliquam dolores facilis nostrum. Sint,
        praesentium architecto ut voluptates id autem unde ipsum odio
tempore
        nulla inventore dolores officiis porro cupiditate fugiat laboriosam,
        nemo, blanditiis consequatur ex. Accusantium laborum quidem dolorum,
        ipsa doloribus officia consequuntur veritatis!
      </p>
    </div>
  </body>
</html>
```

style.css

JavaScript

```
.float-left {
  float: left;
  width: 170px;
  height: 170px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}
.float-right {
  float: right;
  width: 170px;
  height: 270px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}

.para {
  clear: none;
}
```

Output:



clear: right

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <div class="float-left">LEFT</div>
      <div class="float-right">Right</div>
      <p class="para">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Asperiores
        eaque ad nam eum cupiditate dicta corrupti tempore, enim libero
ducimus
        autem inventore numquam ut aliquam dolores facilis nostrum. Sint,
        praesentium architecto ut voluptates id autem unde ipsum odio
tempore
        nulla inventore dolores officiis porro cupiditate fugiat laboriosam,
        nemo, blanditiis consequatur ex. Accusantium laborum quidem dolorum,
```

```
        ipsa doloribus officia consequuntur veritatis!
      </p>
    </div>
  </body>
</html>
```

style.css

JavaScript

```
.float-left {
  float: left;
  width: 170px;
  height: 170px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}
.float-right {
  float: right;
  width: 170px;
  height: 270px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}

.para {
  clear: right;
}
```

Output:

Lorem ipsum dolor sit amet consectetur adipisicing elit. Asperiores eaque ad nam eum cupiditate dicta corrupti tempore, enim libero ducimus autem inventore numquam ut aliquam dolores facilis nostrum. Sint, praesentium architecto ut voluptates id autem unde ipsum odio tempore nulla inventore dolores officiis porro cupiditate fugiat laboriosam, nemo, blanditiis consequatur ex. Accusantium laborum quidem dolorum, ipsa doloribus officia consequuntur veritatis!

clear:both

index.html

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <div class="float-left">LEFT</div>
      <div class="float-right">Right</div>
      <p class="para">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Asperiores
        eaque ad nam eum cupiditate dicta corrupti tempore, enim libero
ducimus
        autem inventore numquam ut aliquam dolores facilis nostrum. Sint,
```

```
        praesentium architecto ut voluptates id autem unde ipsum odio
tempore
        nulla inventore dolores officiis porro cupiditate fugiat laboriosam,
        nemo, blanditiis consequatur ex. Accusantium laborum quidem dolorum,
        ipsa doloribus officia consequuntur veritatis!
      </p>
    </div>
  </body>
</html>
```

style.css

JavaScript

```
.float-left {
  float: left;
  width: 170px;
  height: 170px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}
.float-right {
  float: right;
  width: 170px;
  height: 270px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}

.para {
  clear: both;
}
```

Output:

In the above example, we saw that it gives the same output as **clear:right,** because the right element is bigger than the left element, so if text is cleared from the right element, means cleared from both elements.

Clear: inherit

index.html

```javascript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div>
      <div class="float-left">LEFT</div>
      <div class="float-right">Right</div>
      <p class="para">
```

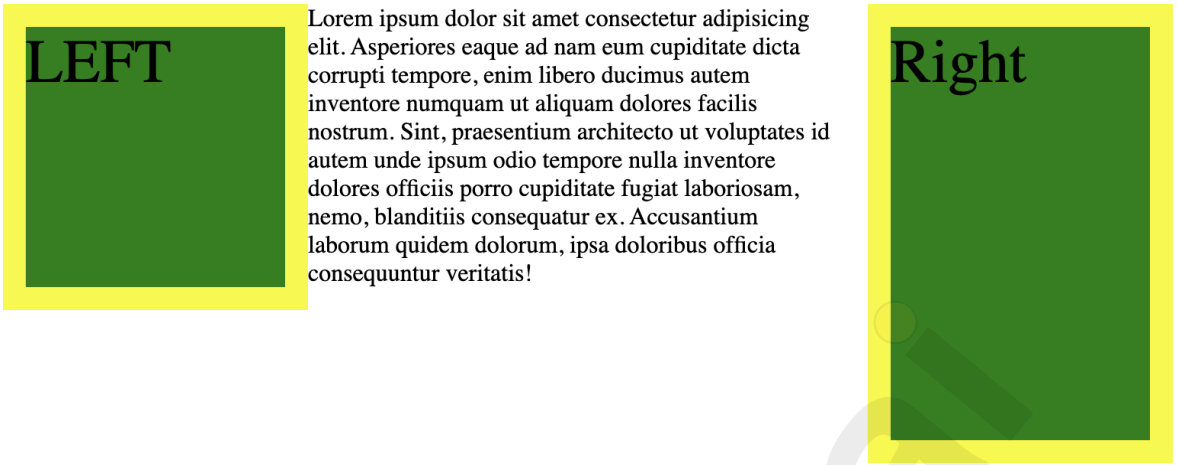
```
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Asperiores
        eaque ad nam eum cupiditate dicta corrupti tempore, enim libero
ducimus
        autem inventore numquam ut aliquam dolores facilis nostrum. Sint,
        praesentium architecto ut voluptates id autem unde ipsum odio
tempore
        nulla inventore dolores officiis porro cupiditate fugiat laboriosam,
        nemo, blanditiis consequatur ex. Accusantium laborum quidem dolorum,
        ipsa doloribus officia consequuntur veritatis!
      </p>
    </div>
  </body>
</html>
```

style.css

JavaScript

```
.float-left {
  float: left;
  width: 170px;
  height: 170px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}
.float-right {
  float: right;
  width: 170px;
  height: 270px;
  margin-left: 15px;
  background-color: green;
  border: 15px solid rgb(248, 248, 0);
  font-size: 40px;
}

.para {
  clear: inherit;
}
```

Output:



In the above example, it is equivalent to none, because parent element has **clear:none** (default).

CSS POSITIONS

CSS positioning is a key part of web design that helps you control where elements appear on a webpage. Using different positioning options, you can place elements exactly where you want them, making it easier to create beautiful and well-organized layouts.

There are five main values from the position property:

1. Static
2. Relative
3. Absolute
4. Fixed
5. Sticky

Why to use CSS Position

1. **Control over element placement:**
   It allows us to control exactly where an element appears on the webpage, giving us better control over the layout.
2. **Positioning relative to other elements:**
   We can use the position properties to place an element based on the location of its parent or nearby elements.
3. **Removing elements from normal flow:**
   The position property lets us move elements out of the normal document flow and place them anywhere on the webpage.
   Example: Modals or pop-ups on websites can be easily positioned using this property.
4. **Overlapping elements:**
   The position property, combined with the **z-index** property, lets us stack elements on top of each other. The element with the higher **z-index** will appear in front of those with a lower **z-index**.
   Example: Pop-up windows (like confirmation boxes) often need to appear above other content on the page.
5. **Positioning relative to the viewport:**
   You can position an element based on the screen's width and height, making sure it stays in the same spot even as the user scrolls.
   Example: Chat support icons that stay fixed in the corner of a website.
6. **Creating a scroll effect:**
   You can make a website's header or footer stay visible at the top or bottom of the screen even while the user scrolls.
7. **Accessibility:**
   By controlling the order and placement of elements, it becomes easier for people using screen readers or keyboard navigation to interact with the webpage.

Position Properties: Top, Left, Right, Bottom

CSS has several properties that help with positioning, such as **top**, **bottom**, **left**, and **right**. These allow you to specify the exact position of an element.

- **Top:** Sets how far the element is from the top of its parent container.
- **Bottom:** Sets how far the element is from the bottom of its parent container.
- **Left:** Sets how far the element is from the left side of its parent container.

- **Right:** Sets how far the element is from the right side of its parent container.

These properties work together with the **position** property to fine-tune the placement of elements on the page. We will learn more about below.

static:

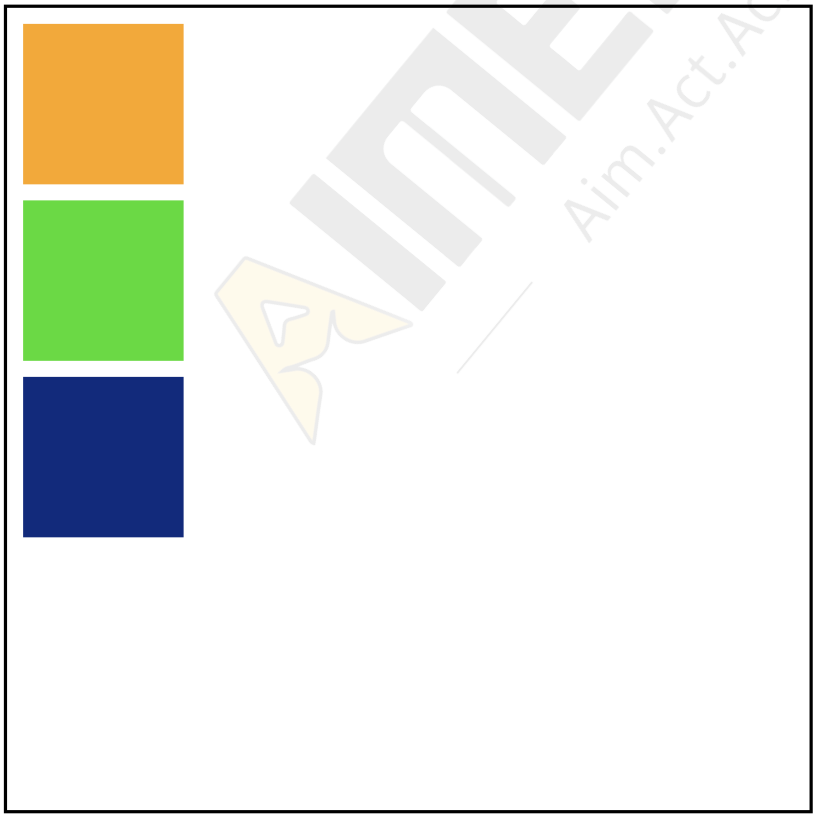
This is the **default position** for all HTML elements. Elements with static positioning are positioned based on the normal document flow, meaning they are placed one after the other in the order they appear in the HTML code.

Let's take an example to understand better:

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .container {
        border: 2px solid black;
        height: 500px;
        width: 500px;
      }
      div {
        margin: 10px;
      }
      .box {
        width: 100px;
        height: 100px;
      }
      .box1 {
        background-color: orange;
      }
      .box2 {
        background-color: rgb(43, 220, 23);
        position: static;
        top: 50px;
```

```
      }

      .box3 {
        background-color: rgb(7, 43, 128);
        position: static;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="box box1"></div>
      <div class="box box2"></div>
      <div class="box box3"></div>
    </div>
  </body>
</html>
```

Output:



As we can see in the above output that the default output. After applying the top property on box 2 which will provide a space from the top of 50px as per code, but has no effect because we are using the static property which is the default one, and the properties like top, bottom, left, and right have no effect in.

Note: static positioned elements cannot have the **z-index, top, left, right, or bottom** properties applied to them.

Relative:

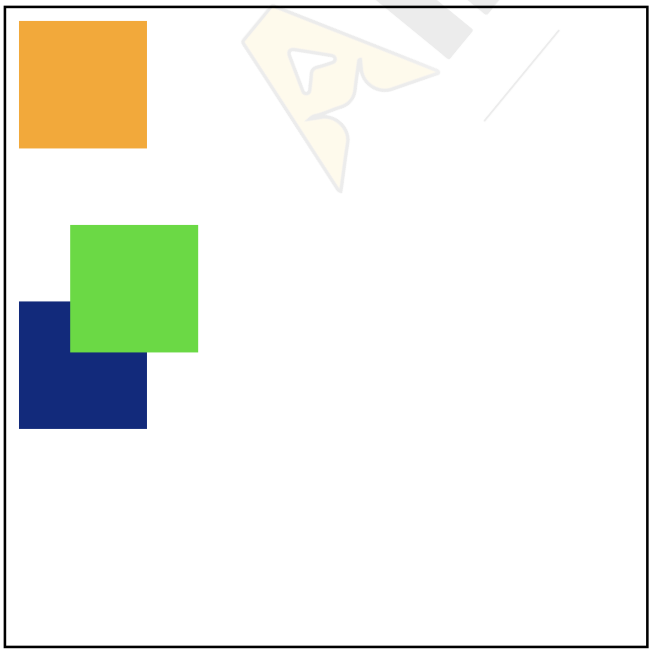
The next type of positioning is **relative positioning.** It works just like static positioning, but you can now use properties like **z-index, top, left, right, and bottom**. If you don't set any of these properties, the element will look the same as a static one. This is because relative elements still follow the normal flow of the page. However, you can shift them by using the **top, left, right, or bottom** properties to move them from their original position.

JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .container {
        border: 2px solid black;
        height: 500px;
        width: 500px;
      }
      div {
        margin: 10px;
      }
      .box {
        width: 100px;
        height: 100px;
      }
```

```
      .box1 {
        background-color: orange;
      }
      .box2 {
        background-color: rgb(43, 220, 23);
        position: relative;
        top: 50px;
        left: 40px;
      }

      .box3 {
        background-color: rgb(7, 43, 128);
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="box box1"></div>
      <div class="box box2"></div>
      <div class="box box3"></div>
    </div>
  </body>
</html>
```

Output:



After applying the position relative, we can see that box2 has moved from its **top** by **50px** and **left** position by **40px,** leaving space at its original position and also not breaking the document flow.

Absolute

With absolute positioning, an element is positioned relative to its nearest positioned ancestor. If there is **no positioned** ancestor, then the element is positioned **relative** to the document body.

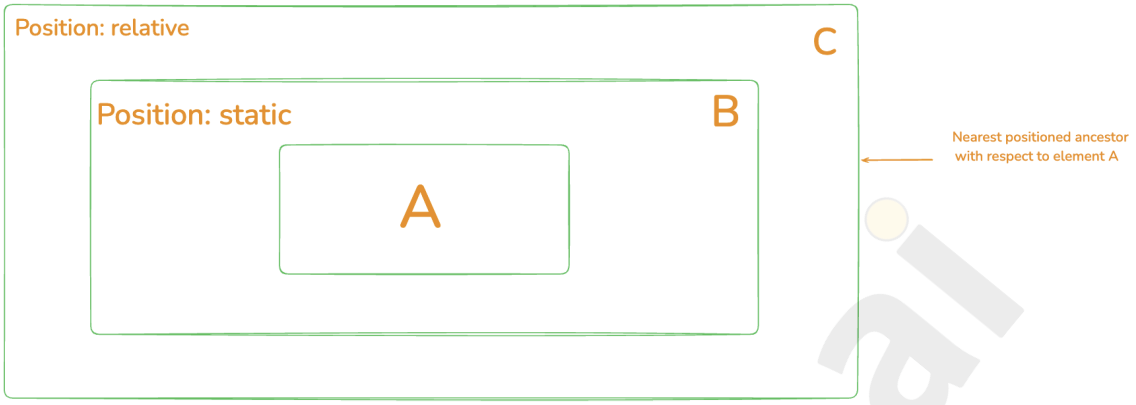
Let's understand the term **nearest positioned ancestor**, We can break it down into two parts: **nearest ancestor** & **positioned ancestor.**

Nearest ancestor: As the name suggests, It is a parent element, which is nearest to the current element.

Positioned ancestor: A parent element that is not positioned static (eg. relative, absolute, fixed, or sticky.).

so, a parent element that is positioned and nearest to the current element is called the nearest positioned ancestor.

Look at the below diagram, element **C** is called the nearest positioned ancestor for element **A.**



Properties of Position Absolute:

- It will break the normal document flow to position the element on the page.
- The properties like top, left, right, bottom, and z-index will have an effect on the element.
- The element will not leave any space in its original position.

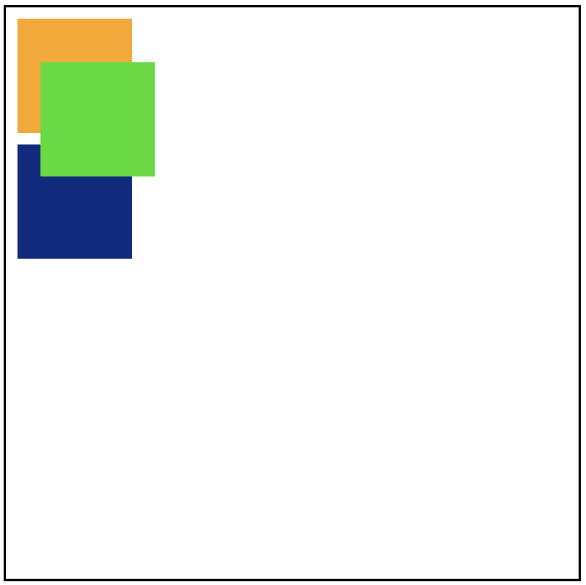
This type of positioning is often used to position elements precisely on a web page.

Let's take an example to understand better:

```javascript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .container {
        border: 2px solid black;
        height: 500px;
```

```
        width: 500px;
      }
      div {
        margin: 10px;
      }
      .box {
        width: 100px;
        height: 100px;
      }
      .box1 {
        background-color: orange;
      }
      .box2 {
        background-color: rgb(43, 220, 23);
        position: absolute;
        top: 50px;
        left: 40px;
      }

      .box3 {
        background-color: rgb(7, 43, 128);
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="box box1"></div>
      <div class="box box2"></div>
      <div class="box box3"></div>
    </div>
  </body>
</html>
```

Output:



As we can see in the output below, the box2 has come out of the flow of the document and is also positioned 40px left and 50px top with respect to the container (positioned ancestor).

Absolute Vs Relative

Absolute	Relative
The element is completely removed from the normal document flow, so it no longer affects the layout of other elements.	The element stays in its normal place within the document flow.
You can place it anywhere on the page using top, left, right, or bottom, but its position is based on the closest positioned parent element (or the whole page if no parent is positioned).	You can move it slightly using top, left, right, or bottom properties, but the space it originally occupied is still reserved.
Example: If you set an element to absolute, it can float over other elements and won't push them aside. It's like you're sticking it somewhere on the page independently.	**Example:** If you move an element 10px to the right, it shifts, but the space where it used to be is still considered part of the layout.

Fixed

Fixed positioning is like absolute positioning, but instead of being placed relative to a parent element, the element is positioned relative to the **viewport** (the browser window). This means the element stays in the same spot, even when the user scrolls the page.

It's commonly used for things like headers, footers, or buttons that need to always be visible.

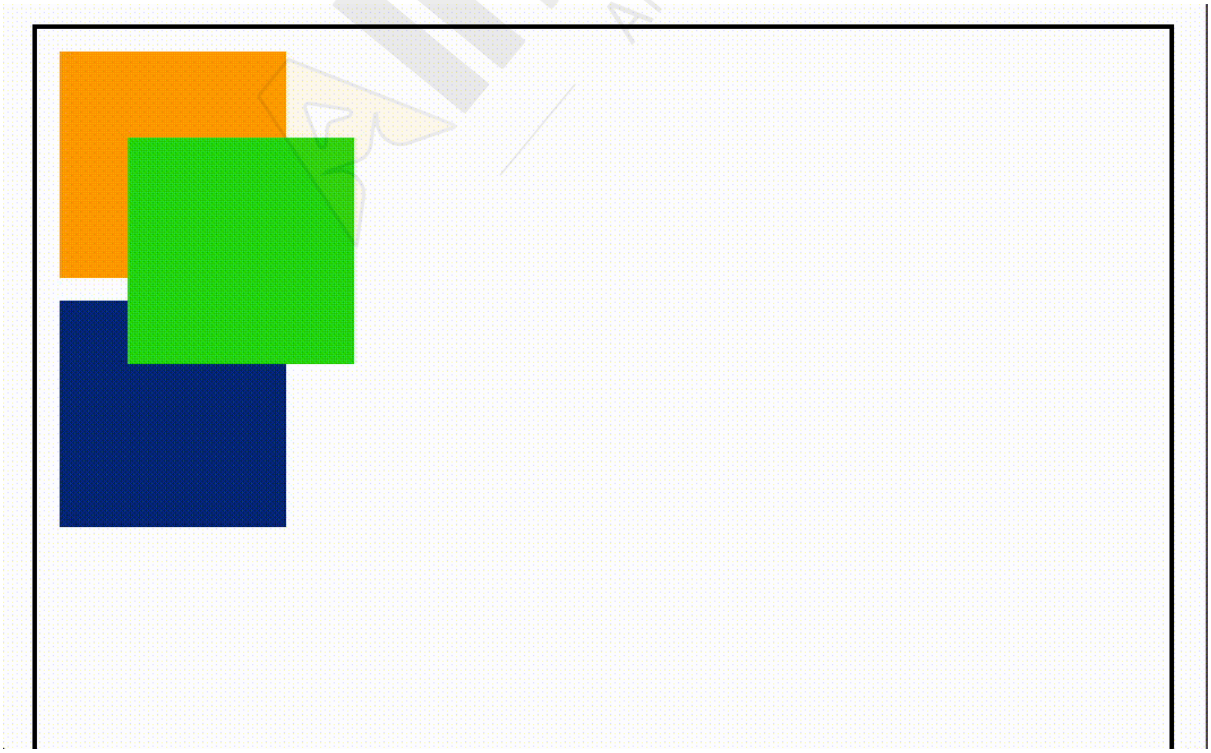
- **Top, left, right, bottom,** and **z-index** can be used to control where the element is placed on the page.
- The element doesn't leave any empty space where it was originally placed.

Example:
If you apply **fixed positioning** to an element (like "box2"), it will stay in that spot on the screen, no matter how far you scroll.

JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .container {
        border: 2px solid black;
        height: 500px;
        width: 500px;
      }
      div {
        margin: 10px;
      }
      .box {
        width: 100px;
        height: 100px;
      }
      .box1 {
        background-color: orange;
        position: relative;
```

```
        overflow: scroll;
      }
      .box2 {
        background-color: rgb(43, 220, 23);
        position: fixed;
        top: 50px;
        left: 50px;
      }

      .box3 {
        background-color: rgb(7, 43, 128);
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="box box1"></div>
      <div class="box box2"></div>
      <div class="box box3"></div>
    </div>
  </body>
</html>
```



In the output of the above code, we can easily see that the box is maintaining its fixed position even if the user is scrolling and other page content is moving.

Note: The overflow property in CSS determines what happens to the content that is too large to fit in an element's box. We are using it here for better visualization of the example and will learn more about it in further classes.

Sticky

Sticky positioning makes an element act like **relative positioning** at first, but when you scroll to a certain point, it "sticks" and behaves like **fixed positioning**. This means the element stays in place as you scroll past it.

It's often used for things like sticky navigation bars or headers that stay visible after scrolling.

Properties of Sticky Positioning:

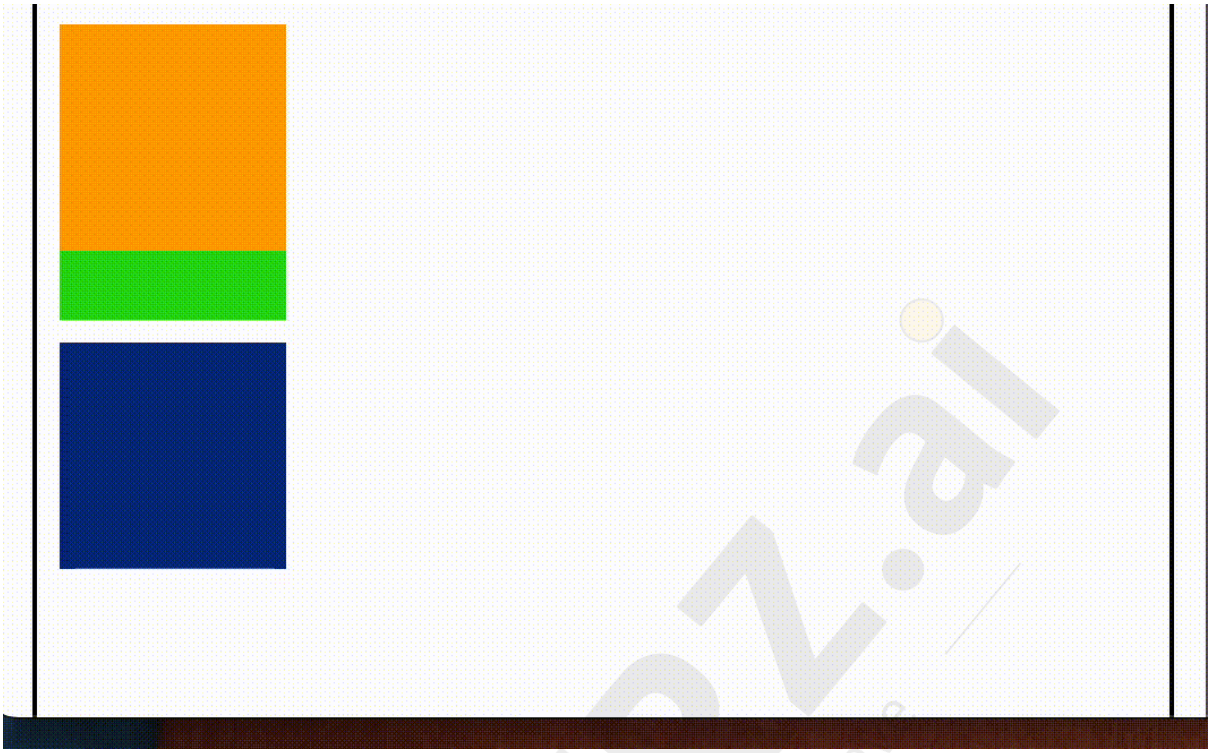
- It won't disrupt the normal flow of the page.
- The element switches between **relative** and **fixed** positioning. It acts like it's in a relative position until you scroll to a certain point, then it sticks to the screen like it's fixed.
- You can use **top**, **left**, **right**, **bottom**, and **z-index** to control its position.

Example: A sticky header stays at the top of the page as you scroll down.

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .container {
        border: 2px solid black;
        height: 500px;
```

```
        width: 500px;
      }
      div {
        margin: 10px;
      }
      .box {
        width: 100px;
        height: 100px;
      }
      .box1 {
        background-color: orange;
        position: sticky;
        top: 10px;
      }
      .box2 {
        background-color: rgb(43, 220, 23);
      }

      .box3 {
        background-color: rgb(7, 43, 128);
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="box box1"></div>
      <div class="box box2"></div>
      <div class="box box3"></div>
    </div>
  </body>
</html>
```

Output:



Fixed vs Sticky

Fixed	Sticky
The element is always stuck in one spot relative to the viewport (browser window).	The element acts normally (like it's relatively positioned) until you scroll to a certain point.
It stays in the same place, no matter how far you scroll.	Once you scroll past that point, it "sticks" and behaves like fixed positioning, staying visible as you keep scrolling.
Commonly used for things like headers or buttons that must always be visible.	Often used for sticky navigation bars that stick at the top of the page after you scroll.

CSS Overflow

When you create a website, sometimes the content inside a box (like a div or section) can be too big. This can make the page look messy or cause a horizontal scrollbar to appear, which is usually not what we want.

The **overflow** property in CSS helps us control what happens to this extra content. It tells the browser how to handle content that doesn't fit in its box.

How to Use the Overflow Property

You can use the overflow property to set how content behaves in either horizontal (x-axis) or vertical (y-axis) directions, or both. Here's how:

1. **For horizontal direction only:**

```
Unset
overflow-x: value; /* Controls overflow in the horizontal direction */
```

2. **For vertical direction only:**

```
Unset
overflow-y: value; /* Controls overflow in the vertical direction */
```

3. For both directions at once:

```
JavaScript
overflow: value; /* Controls overflow in both directions */
```

Possible Values for Overflow

1. **visible:**

   The content is visible even if it overflows the box.

   Example: If you have a box with this value, any extra content will just stick out.

2. **hidden:**

   The overflowing content is not visible at all.

   Example: Any content that goes beyond the box will be cut off and not shown.

3. **scroll:**

   A scrollbar is always shown, regardless of whether there is overflowing content.

   Example: You will see a scrollbar even if the content fits perfectly in the box.

4. **auto:**

   A scrollbar is shown only if there is overflowing content.

   Example: If the content fits, no scrollbar appears. If it doesn't fit, a scrollbar will show up.

1. **Overflow: visible**

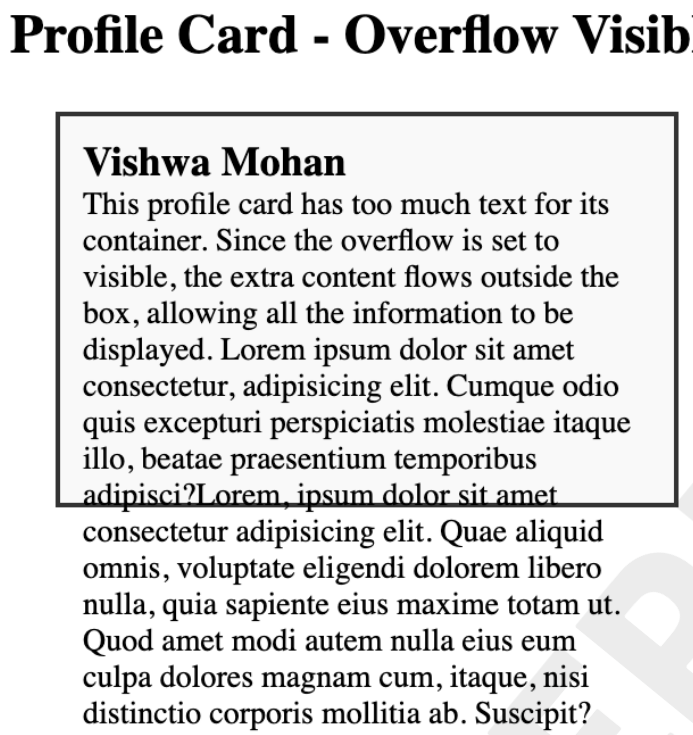
By default the overflow is visible, meaning that it is not clipped and it renders outside the element's box.

Let's take an example to understand better:

```
JavaScript
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Profile Card</title>
    <style>
      .profile-card {
        width: 250px;
        height: 150px;
        border: 2px solid #333;
        background-color: #f9f9f9;
        overflow: visible;
        padding: 10px;
        margin: 20px;
      }

      .profile-content {
        font-size: 14px;
      }

      .name {
        font-weight: bold;
        font-size: 18px;
      }
    </style>
  </head>
  <body>
    <h2>Profile Card - Overflow Visible</h2>
    <div class="profile-card">
      <div class="name">Vishwa Mohan</div>
      <div class="profile-content">
        This profile card has too much text for its container. Since the
        overflow is set to visible, the extra content flows outside the box,
        allowing all the information to be displayed. Lorem ipsum dolor sit
amet
        consectetur, adipisicing elit. Cumque odio quis excepturi
perspiciatis
        molestiae itaque illo, beatae praesentium temporibus adipisci?Lorem,
        ipsum dolor sit amet consectetur adipisicing elit. Quae aliquid
omnis,
        voluptate eligendi dolorem libero nulla, quia sapiente eius maxime
totam
        ut. Quod amet modi autem nulla eius eum culpa dolores magnam cum,
        itaque, nisi distinctio corporis mollitia ab. Suscipit?
      </div>
    </div>
  </body>
</html>
```

Output:



In the above example , the profile card contains more words than its container can hold and Overflow is set to visible, so the extra content spills out of the box.

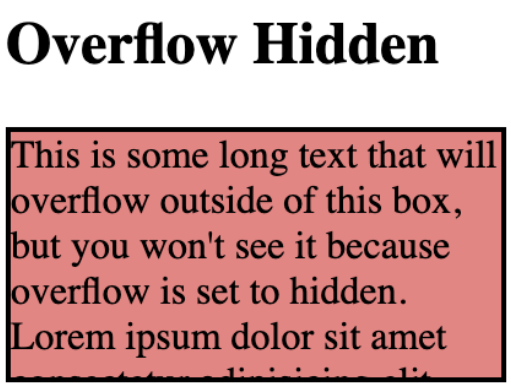
2. Overflow:hidden

When overflow: hidden; is used, content that goes beyond the container's size is cut off and won't be seen outside the box.

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

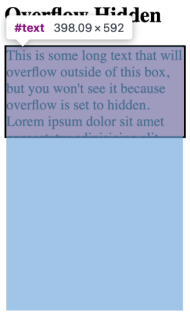
```
    <title>Overflow Hidden Example</title>
    <style>
      .container {
        width: 200px;
        height: 100px;
        border: 2px solid black;
        overflow: hidden;
        background-color: lightcoral;
      }
    </style>
  </head>
  <body>
    <h2>Overflow Hidden</h2>
    <div class="container">
      This is some long text that will overflow outside of this box, but you
      won't see it because overflow is set to hidden. Lorem ipsum dolor sit
amet
      consectetur adipisicing elit. Distinctio tempore id facere pariatur
aut,
      minima fugiat, hic amet itaque corporis maxime sunt nihil soluta nam.
Sed,
      autem reiciendis, dolores sapiente sunt ea iusto beatae, suscipit
      accusantium quos ullam? Ad repellat ipsum in quasi voluptatem non
dolores
      cum dolor incidunt velit.
    </div>
  </body>
</html>
```

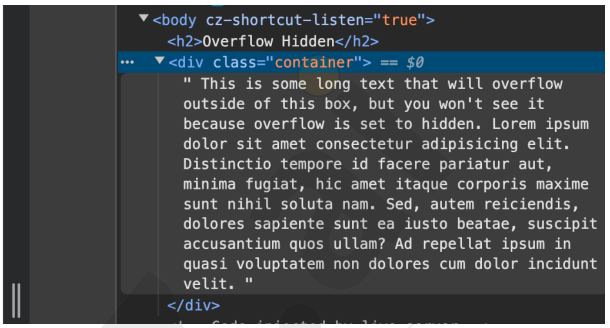
Output:



As we can see, the text that was supposed to overflow is cut off. And to show you that the text is still present in the DOM, I am attaching another screenshot from the browsers element tab

Output:





3. Overflow: scroll

When **overflow: scroll;** is applied, scrollbars are added to the container, allowing users to scroll through any content that goes beyond the container's size, even if the content doesn't overflow.

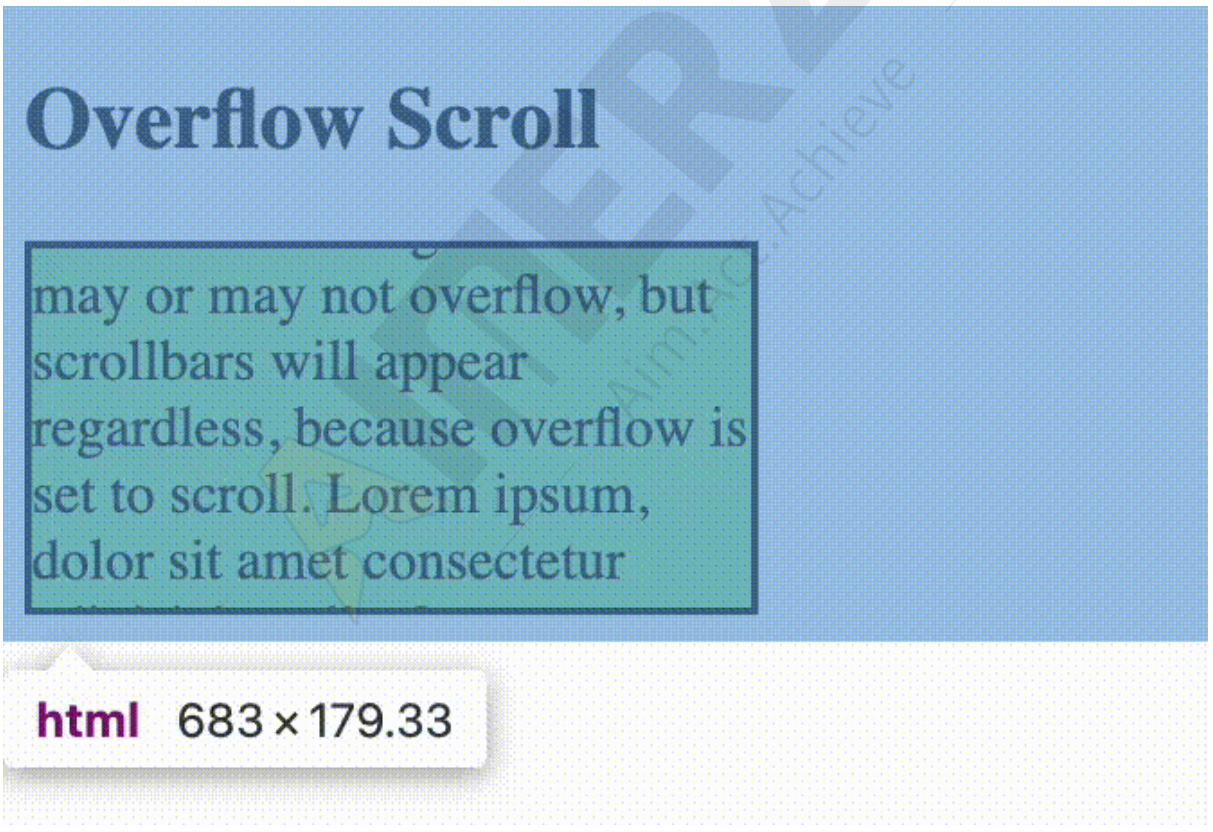
Example:

JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Overflow Scroll Example</title>
    <style>
      .container {
        width: 200px;
        height: 100px;
        border: 2px solid black;
        overflow: scroll;
        background-color: lightgreen;
      }
```

```
    </style>
  </head>
  <body>
    <h2>Overflow Scroll</h2>
    <div class="container">
      This is some long text that may or may not overflow, but scrollbars
will
      appear regardless, because overflow is set to scroll.
    </div>
  </body>
</html>
```

Output:



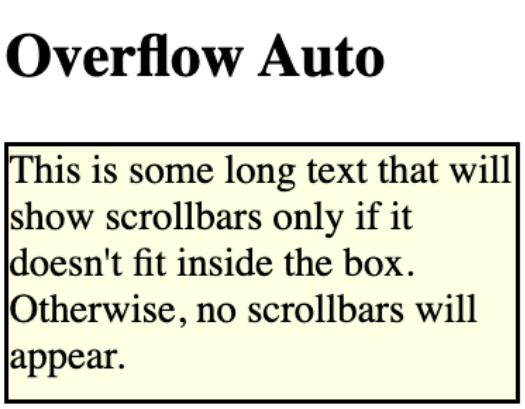
4. overflow: auto;

When overflow: auto; is applied, scrollbars will appear only if the content exceeds the container's size. If the content fits within the container, no scrollbars are shown.

Ex:

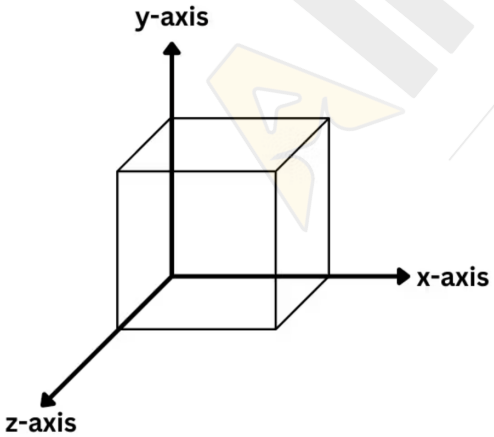
```JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Overflow Auto Example</title>
    <style>
        .container {
            width: 200px;
            height: 100px;
            border: 2px solid black;
            overflow: auto;
            background-color: lightyellow;
        }
    </style>
</head>
<body>
    <h2>Overflow Auto</h2>
    <div class="container">
        This is some long text that will show scrollbars only if it doesn't
fit inside the box. Otherwise, no scrollbars will appear.
    </div>
</body>
</html>
```

Output:



Z-index

The z-index property in CSS helps determine the order of overlapping elements on a webpage, controlling which element appears on top when they stack. This "stacking" happens along the z-axis (the third dimension, coming out from the screen).



Why is it important?

Imagine you're placing pieces of paper on a table. You can layer one on top of another. In the web, when elements (like images or boxes) overlap, z-index helps decide which one sits on top and which one stays behind.

How does `z-index` work?

- Each element can be assigned a z-index value, which can be a positive number, negative number, or zero.
- Elements with higher z-index numbers will appear on top of those with lower numbers.
- If two elements have the same z-index, the one that appears later in the HTML code will be on top.
- For z-index to work, the element must have a position other than the default static (such as relative, absolute, fixed, or sticky).
- **If an element is not positioned (default static), the z-index won't apply.**

Syntax of `z-index`:

```
JavaScript
z-index: auto | number | initial | inherit;
```

- **auto:** This is the default and it means the element will follow its parent's stacking order.
- **number:** You can assign a number like 1, 2, -1, etc. Higher numbers sit on top. You can even use negative values.
- **initial:** Resets the value to the default.
- **inherit:** Inherits the z-index from the parent element.

The z-index values can be **positive, negative,** or **zero.** Elements with **higher z-index** values will always be on **top** of the elements with lower z-index values.

Now, let's take a look at some examples to understand z-index.

Example-1:

index.html

```
JavaScript
<div class="box box1"></div>
<div class="box box2"></div>
```

Style.css

```
JavaScript
.box {
  width: 100px;
  height: 100px;
  position: absolute;
}
.box1 {
  background-color: greenyellow;
  z-index: 2;
  top: 20px;
  left: 20px;
}
.box2 {
  background-color: goldenrod;
  z-index: 1;
}
```

Output:

In the above example, **greenyellow box** (z-index:2) will be on top of the **goldenrod box(** z-index:1)

Example-2: (Same z-index)

index.html

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="box box1"></div>
    <div class="box box2"></div>
  </body>
</html>
```

style.css

```JavaScript
.box {
  width: 100px;
  height: 100px;
  position: absolute;
```

```
}
.box1 {
  background-color: greenyellow;
  z-index: 2;
  top: 20px;
  left: 20px;
}
.box2 {
  background-color: goldenrod;
  z-index: 2;
}
```

Output:

In the above example, both have the same z-index value(z-index:2),the one that comes later in the HTML code (in this case, the goldenrod box) will be on top.

Example-3: Negative z-index

index.html

JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
```

```
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="box box1"></div>
    <div class="box box2"></div>
  </body>
</html>
```

Style.css

JavaScript

```
.box {
  width: 100px;
  height: 100px;
  position: absolute;
}
.box1 {
  background-color: greenyellow;
  z-index: -1;
  top: 20px;
  left: 20px;
}
.box2 {
  background-color: goldenrod;
  z-index: -2;
}
```

Output:

In the above example , both boxes will sit behind other elements on the page because they have negative z-index values, but the red box (-1) will still be on top of the blue box (-2) since -1 is higher than -2.

THANK YOU

AIMERZ.ai
Aim.Act.Achieve