AIMERZ.ai
Aim.Act.Achieve

CSS

NOTES

Advance CSS -2

Transform

CSS transforms enable you to manipulate the appearance and position of elements without changing their layout in the HTML document. This property is essential for creating interactive and visually engaging web pages.

Types of Transforms in CSS

2D Transform

2D transformations enable you to manipulate elements on a **flat surface**, such as a web page. This includes changing their position (moving them around), rotation (turning them at an angle), and scaling (changing their size).
These transformations operate within a two-dimensional space, meaning they don't involve depth like 3D transformations do. In simpler terms, with 2D transformations, you can make elements look different by shifting them, spinning them, or resizing them, all while keeping them on the same flat plane.

1. **Translate:**

   The translate property is used to move the element alone on the x-axis and y-axis.

Syntax:

```
Unset
translate(x, y)
```

The x and y parameters specify the distance that the element should be moved along the X and Y axis, respectively. They can be specified in various units, such as “px”, “em” or “%”.

By using translateX() we can move the element on the x-axis and by using translateY() we can move the element on the y-axis.

Let's take some example to understand better how to use translate in css():

1. **Move the element 50px to the right and 25px down**

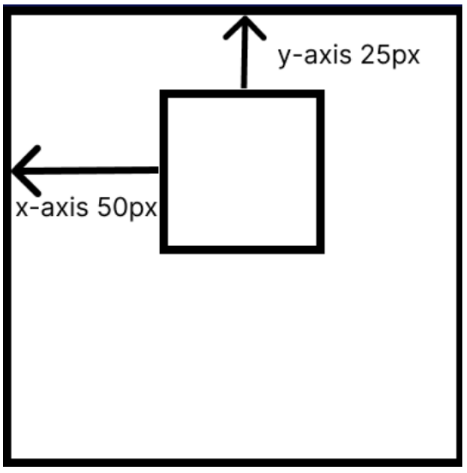
index.html

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

style.css

```
JavaScript
.box {
  border: solid;
  height: 50px;
  width: 50px;
  transform: translate(50px, 25px);
}
```

Output:



2) Move the element 60 pixels to the right using **translateX()**:

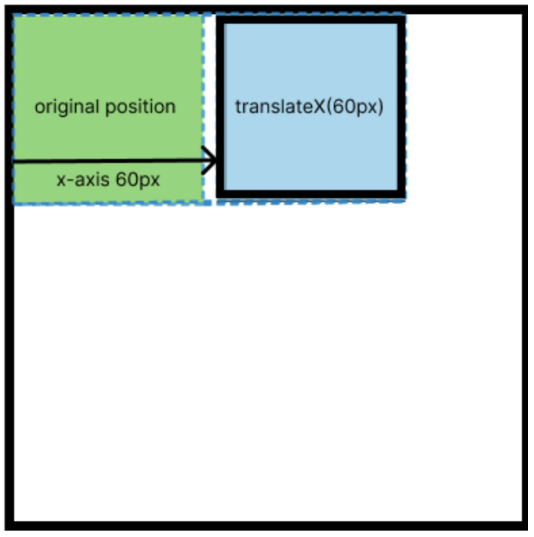
index.html

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

Style.css

```
JavaScript
.box {
  border: solid;
  height: 50px;
  width: 50px;
  transform: translateX(60px);
}
```

Output: With some required information for better understanding



3. Move the element 70 pixels to the bottom using **translateY**():

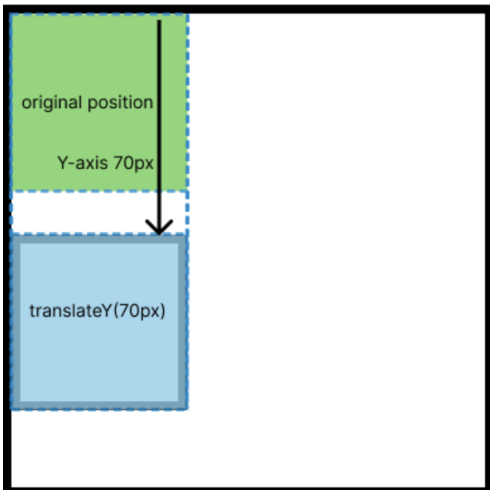
index.html

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

Style.css

```
JavaScript
.box {
  border: solid;
  height: 50px;
  width: 50px;
  transform: translateY(70px);
}
```

Output:



2. scale:

It is used to change the width and height of an element.

Syntax:

```
Unset
transform: scale(x, y);
```

where x and y are the scaling factors. A value of **1** represents the original size of the element. Values greater than 1 will scale the element up, while values between **0** and **1** will scale the element down.

By using **scaleX()** we can change the width of an element and by using **scaleY()** we can change the height of an element.

Let's take some example to understand how to use scale():

1. Scale an element to 120% of its original size in the horizontal direction and 80% of its original size in the vertical direction.

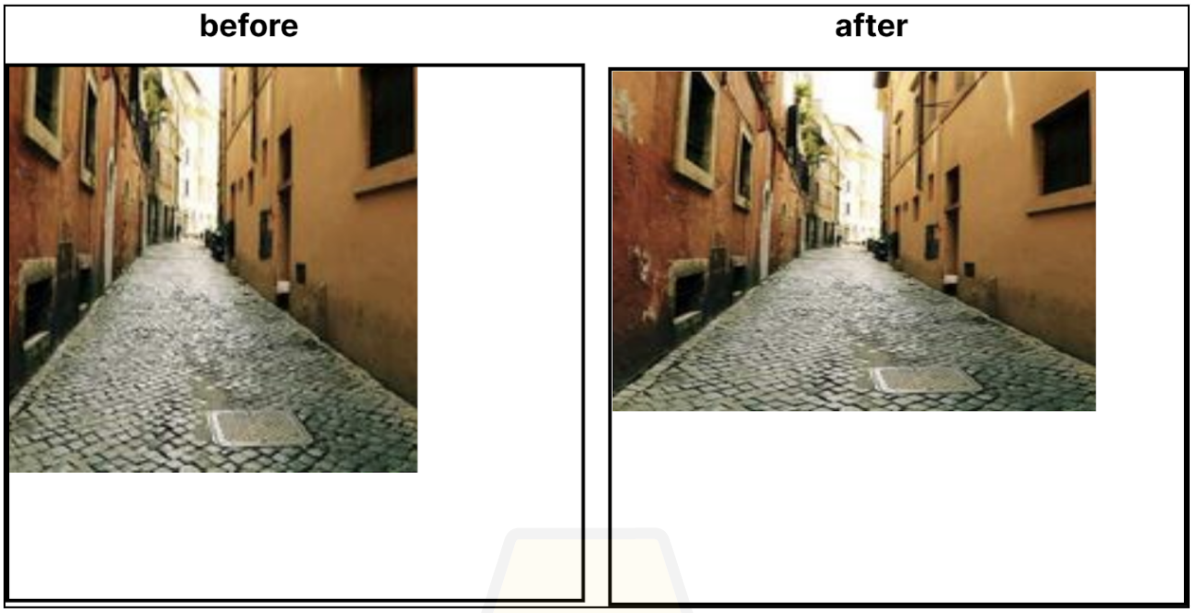
index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="./style.css" />
  <title>Document</title>
 </head>
 <body>
  <img class="image" src="https://picsum.photos/200" alt="image" />
 </body>
</html>
```

Style.css

```
JavaScript
.image{  transform: scale(1.2, 0.8); }
```

Output:



2. Scale an element to 150% of its original size in both the horizontal and vertical directions .

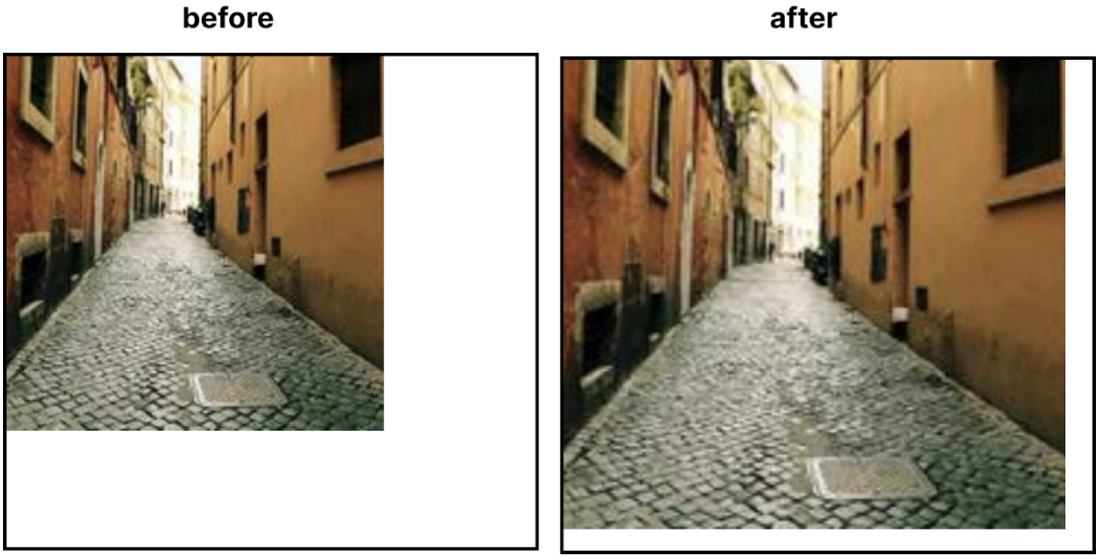
index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="./style.css" />
    <title>Document</title>
  </head>
  <body>
    <img class="image" src="https://picsum.photos/200" alt="image" />
  </body>
</html>
```

style.css

```
Unset
.image{  transform: scale(1.5); }
```

Output:



3) Increase the height and decrease the width of elements using **scaleY()** and **scaleX()**:

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="./style.css" />
    <title>Document</title>
  </head>
  <body>
    <img class="image" src="https://picsum.photos/200" alt="image" />
  </body>
</html>
```

style.css

```
JavaScript
.image{
    transform: scaleX(1.2);
    transform: scaleX(0.5);
}
```

Output:

before

after

3. **Rotate:**

It is used to rotate the element on the basis of an angle.

Syntax:

```
JavaScript
transform: rotate(angle);
```

where **angle** is the amount of rotation in degrees. **Positive values** rotate the element **clockwise**, while **negative values** rotate it **counterclockwise.**

Let's take some example to understand how to use rotate():

index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="./style.css" />
    <title>Document</title>
  </head>
  <body>
    <img class="image" src="https://picsum.photos/200" alt="image" />
  </body>
</html>
```

Note: We will use the same HTML code for all rotate() examples.

1. Rotate an element 45 degrees clockwise:

style.css

```
JavaScript
.image { transform: rotate(45deg); }
```

Output:

2. Rotate an element 45 degrees anticlockwise:

style.css

```
JavaScript
.image {  transform: rotate(-45deg); }
```

Output:

4. Skew:

It specifies the skew transformation along the X and Y axis corresponding to the skew angles.

Syntax:

```
Unset
transform: skew(x-angle, y-angle);
```

where **x-angle** and **y-angle** are the **angles of skew in degrees**. **Positive values skew** the element **in the direction of the positive axis**, while **negative values skew** it in the **opposite direction**.

By using **skewX()** we can skew the element along the x-axis and by using **skewY()** we can skew the element along the x-axis**.**

Let us take some examples of how using skew() in CSS

Index.html

```
JavaScript
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./style.css">
    <title>Document</title>
</head>
<body>
    <img class="image" src="https://picsum.photos/200" alt="image"/>
</body>
</html>
```

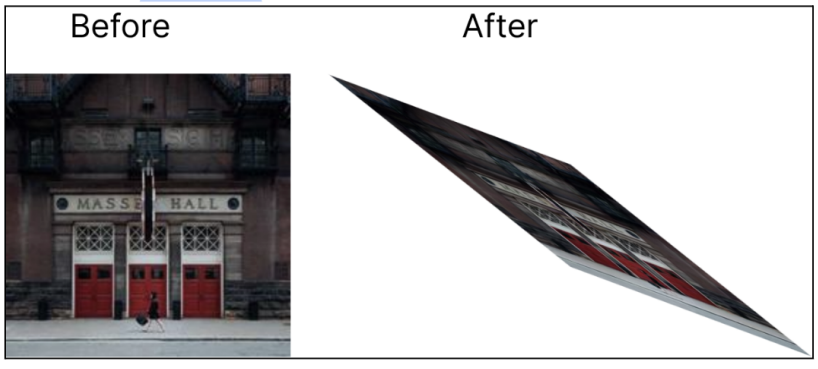
***Note*:** We will use the same HTML code for all skew examples.

1. skew an element 45 degrees in the X direction and 25 degrees in the Y direction.

style.css

```
JavaScript
.image {  transform : skew(45deg,25deg) }
```

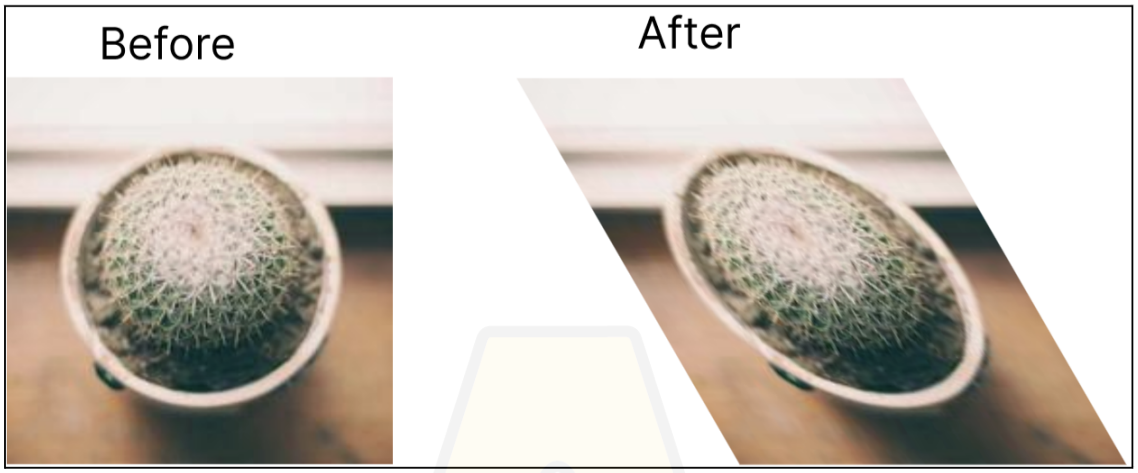
output:



2) Skew an element 30 degrees in the X direction using **skewX()**:

style.css

```
JavaScript
.image {      transform: skewX(30deg)  }
```



3) Skew an element 45 degrees in the Y direction using **skewY()**:

style.css

```
JavaScript
image{
       transform: skewY(45deg)
}
```

Output:

5. Matrix:

The matrix() defines a homogeneous 2D transformation matrix.
It takes six parameters, containing mathematical functions that allow you to rotate, scale, move (translate), and skew elements.

syntex:

```
Unset
transform: matrix(a, b, c, d, tx, ty);

// matrix(scaleX(), skewY(), skewX(), scaleY(), translateX(),

translateY())

//
```

where **a**, **b**, **c**, **d**, **tx**, and **ty** are the values of the matrix elements. These values represent the following transformation matrix:

```
Unset
| a  c  tx |
| b  d  ty |
| 0  0  1  |
```

Here's an explanation of what each matrix element does:

- “a” represents the horizontal scaling of the element.
- “b” represents the vertical skewing of the element.
- “c” represents the horizontal skewing of the element.
- “d” represents the vertical scaling of the element.
- “tx” represents the horizontal translation of the element.
- “ty” represents the vertical translation of the element.

Here are some examples of how to use matrix() in CSS:

1) Scale the element in horizontal and vertical directions:

style.css

```
Unset
 .image{
      transform: matrix(2, 0, 0, 2, 0, 0);
}
```

output:

3D Transform

We can perform 3D transformations to manipulate the **position**, **rotation**, and **scaling** of elements **in a three-dimensional space** on a web page. These transformations allow us to create interactive and visually appealing 3D effects without the need for complex JavaScript or external libraries.

Lets see Some common 3D transform properties with simple examples.

1. rotateX()

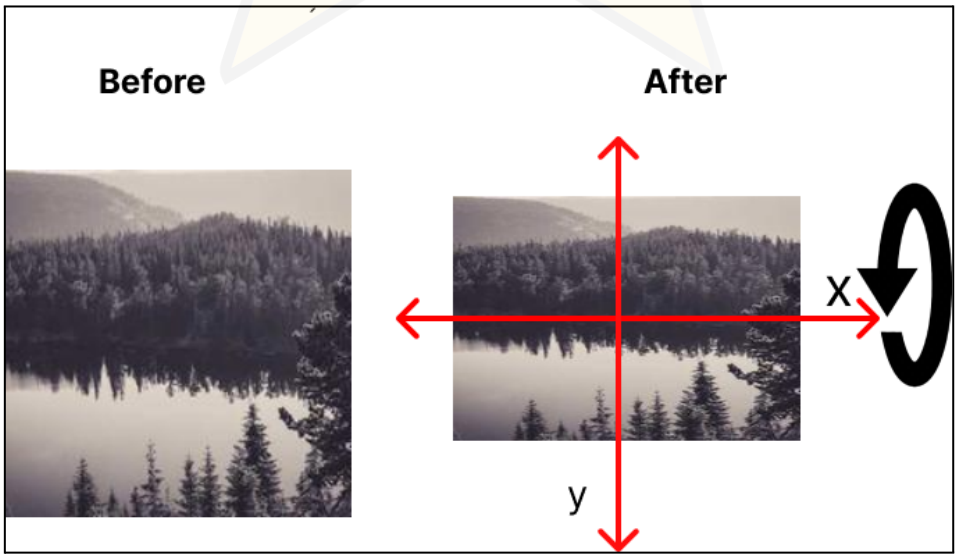
The rotateX() transformation in CSS allows you to rotate an element around the X-axis in 3D space. This rotation creates a tilting or flipping effect along the horizontal axis.

Rotate the element for its pivot in x direction.

style.css

```
Unset
 transform: rotateX(40deg);
```

Browser output



2. rotateY():

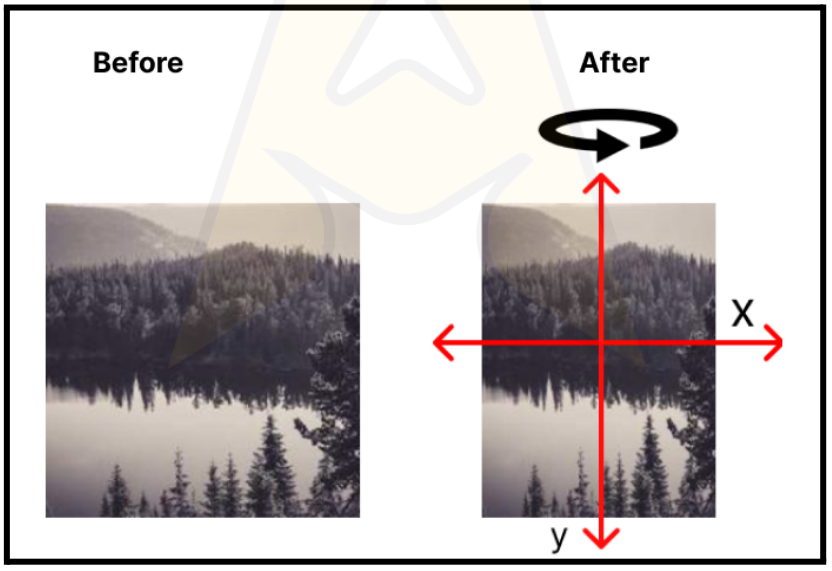
The rotateY() transformation in CSS allows you to rotate an element around the Y-axis in 3D space. This rotation creates a horizontal flipping or spinning effect.

Rotate the element for its pivot in y direction.

style.css

```
Unset
 .image{
      transform: rotateY(40deg);
}
```

output:



3. rotateZ():

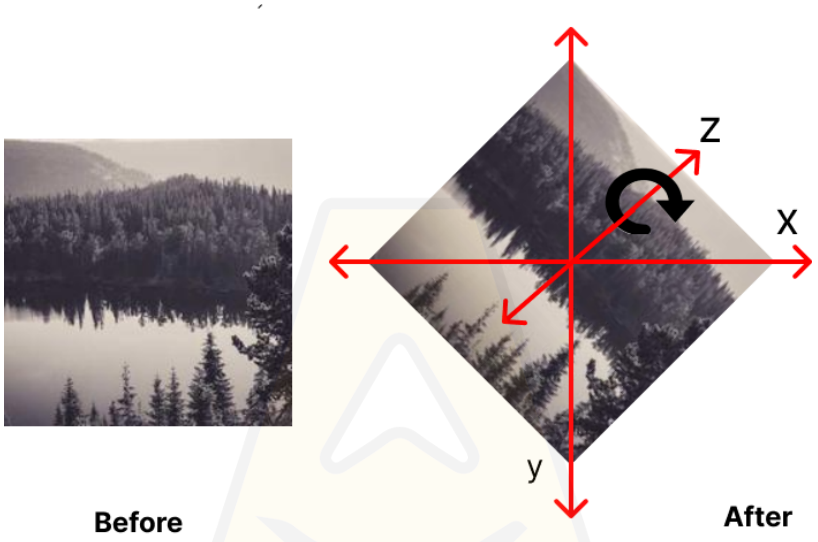
The rotateZ() transformation in CSS allows you to rotate an element around the Z-axis in 3D space. This rotation creates a twisting or spinning effect, similar to turning a wheel on its axis.

Rotate the element for its pivot in z direction.

style.css

```
Unset
 .image{
      transform: rotateY(40deg);
}
```

output:



2D transformations work on a flat surface, like moving objects on a piece of paper. 3D transformations add depth and allow objects to move, rotate, and change size in a three-dimensional space, like virtual objects in a 3D game.

Transition

What is Transition?
CSS transitions enable you to control the speed of animation as you modify CSS attributes.

CSS Transition consists of,

- When animation will trigger
- When animation will start (delay)
- Duration of transition
- How will transitions run?

Transition Properties
CSS transitions allow you to change the property values of an element smoothly over a specified duration of time.

Here are the main CSS transition properties,

transition-property: Specifies which properties will be transitioned. You can specify a comma-separated list of properties, or use the value **all** to transition all properties.

```
Unset

transition-property: none;
transition-property: color;
transition-property: width;
transition-property: all;
```

transition-duration: specifies the duration of the transition in seconds or milliseconds.

```javascript
transition-duration: 6s;
transition-duration: 120ms;
```

transition-timing-function: specifies the timing function used for the transition. There are several predefined timing functions, such as **ease-in**, **ease-out**, **ease-in-out**, and **linear**.

```unset
transition-timing-function: ease;
transition-timing-function: ease-in;
transition-timing-function: ease-out;
transition-timing-function: ease-in-out;
```

transition-delay: specifies the delay before the transition starts in seconds or milliseconds.

```javascript
transition-delay: 3s;
transition-delay: 3000ms;
```

Transition Shorthand

We can define transitions, in one line using the below syntax.

```unset
transition: property_name duration easing_function delay
```

Examples

Let's try to understand transition using examples,

Example 1: Transition margin-top of button on hover.

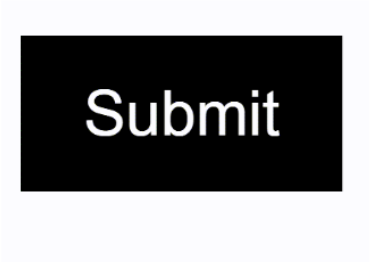
JavaScript

```
<button>Submit</button>

button {
      margin-top: 0px;
      transition: margin-top 0.5s ease-in-out;
      padding: 10px 15px;
      background-color: black;
      color: white;
      border: none;
      font-size: 15px;
}

button:hover {
      margin-top: 5px;
}
```

Output:



Example 2 : Transition background color of links on hover.

JavaScript

```
<nav>
   <a href="#">Home</a>
   <a href="#">About</a>
   <a href="#">Pricing</a>
   <a href="#">Events</a>
</nav>

nav {
   display: flex;
   gap: 0.5rem;
}

a {
  flex: 1;
  background-color: #333;
  color: #fff;
  border: 1px solid;
  padding: 0.5rem;
  text-align: center;
  text-decoration: none;
  transition: all 0.5s ease-out;
}

a:hover, a:focus {
   background-color: #fff;
   color: #445ed1;
}
```

Output

Example 3: Transition on changing width of circle.

JavaScript

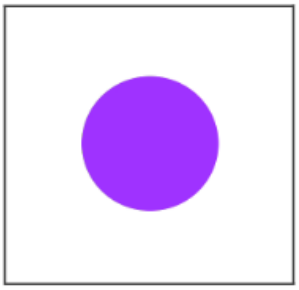
```
<div class="elem"></div>

.elem {
   background: blueviolet;
   width: 150px;
   height: 150px;
   transition: all 500ms ease-in-out;
   border-radius: 75px;
   margin-left: 100px;
   margin-top: 100px;
}

.elem:hover {
  transform: scale(2);
}
```

Output:

Initial :



On hover:

Filter

What is a filter?

The CSS property called **filter** is used to apply various graphical effects such as **color shift** or **blur** to an element. Typically, filters are employed to modify how images, backgrounds, and borders are displayed.

Syntax of filter

To apply filters, we use the following syntax,

```
JavaScript
filter: <filter-function>
```

where filter-functions are predefined functions used to apply various visual effects to elements on a web page.

In this lecture, we will study some common filter effects through examples.

Common Filter functions

1. **blur():** Applies a Gaussian blur to the input image

Syntax

```
JavaScript
filter:blur(radius)
```

Example:-

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Filter</title>
  </head>
  <style>
    div {
      border: 1px solid black;
      width: 200px;
      height: 200px;
    }

    img {
      filter: blur(10px);
    }
  </style>
  <body>
    <div>
     <img src="./assets/css.png" alt="image" width="100%"
     />
    </div>
  </body>
</html>
```

Output:-

Before filter	After filter
	

2. **brightness() :** Making the image appear brighter or darker.

Syntax:

```
JavaScript
filter:brightness(amount)
```

Example:

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Filter</title>
  </head>
  <style>
```

```
    div {
      border: 1px solid black;
      width: 200px;
      height: 200px;
    }
    img {
      filter: brightness(0.4);
    }
  </style>
  <body>
    <div>
     <img src="./assets/css.png" alt="image" width="100%"
     />
    </div>
  </body>
</html>
```

Output:-

Before filter	After filter
	

3. **grayscale() :** Converts the input image to grayscale

Syntax:

```
JavaScript
filter:grayscale(amount)
```

Example:-

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Filter</title>
  </head>
  <style>
    div {
      border: 1px solid black;
      width: 200px;
      height: 200px;
    }

    img {
      filter: grayscale(0.8);
    }
  </style>
  <body>
    <div>
      <img src="./assets/css.png" alt="image" width="100%" />
    </div>
  </body>
</html>
```

Output:-

Before filter	After filter
	

4. **hue-rotate() :** Rotates the hue of an element and its contents.

Syntax:

```
JavaScript
filter:hue-rotate(angle)
```

Example:

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Filter</title>
  </head>
  <style>
    div {
      border: 1px solid black;
      width: 200px;
      height: 200px;
    }

    img {
      filter: hue-rotate(90deg);
    }
  </style>
  <body>
    <div>
      <img src="./assets/css.png" alt="image" width="100%" />
    </div>
  </body>
</html>
```

Output:-

Before filter	After filter
	

5. **drop-shadow() :** Applies a drop shadow effect to the input image or text.

Syntax:

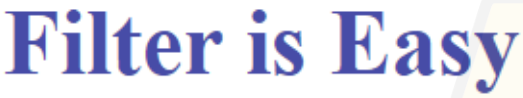
```
JavaScript
drop-shadow(offset-x offset-y blur-radius color)
```

Example

```
JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Filter</title>
  </head>
  <style>
    h1 {
      color: rgb(71, 76, 167);
      filter: drop-shadow(30px 30px 0 rgb(103, 97, 97));
    }
  </style>
  <body>
```

```
    <div>
      <h1>Filter is Easy</h1>
    </div>
  </body>
</html>
```

Output:-

Before filter	After filter
	

Combining Multiple Filters

We can combine multiple filters, to give more visual effects.

Syntax

```
JavaScript
filter: <filter-function-1> <filter-function-2>
```

Example

```JavaScript
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Filter</title>
  </head>
  <style>
    img {
      filter: drop-shadow(30px 10px 2px #3fb1e2)
hue-rotate(90deg)
        drop-shadow(30px 10px 2px #3fb1e2);
    }
  </style>
  <body>
    <div>
      <img src="./assets/css.png" alt="image" width="200px" />
    </div>
  </body>
</html>
```

Output:-

If you have noticed, now there are two shadows, with two different colors.

THANK YOU

AIMERZ.ai
Aim.Act.Achieve