Lesson:

Transform

PW SKILLS

List of Concepts Involved:

- **2D**
  a. translate()
  b. scale()
  c. rotate()
  d. skew()
  e. matrix()
- **3D**
  a. rotateX()
  b. rotateY()
  c. rotateZ()

In CSS, a transform is a property that allows you to change the appearance and position of an element without altering its original layout in the HTML document. It enables you to apply various visual effects and animations to elements on a web page, making it more interactive and engaging.

Types of Transforms in CSS

2D Transform

2D transformations allow you to manipulate the position, rotation, and scaling of elements in a two-dimensional space on a web page. Unlike 3D transformations, which work in a three-dimensional space and add depth, 2D transformations only affect elements on a flat plane.

a. translate()

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

Syntex:

```
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.

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

- Move the element 50 pixels to the right and 25 pixels down:

Index.html:

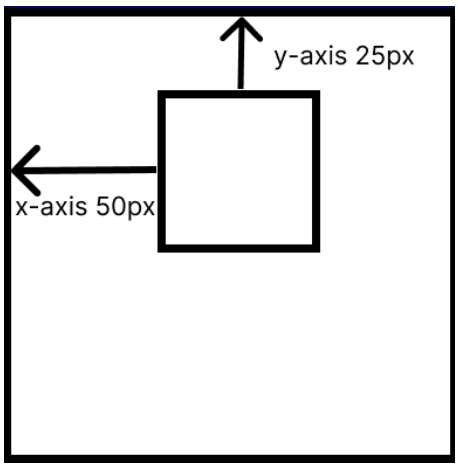
```
Unset
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <link rel="stylesheet" href="./style.css" />
    <title>Document</title>
  </head>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

Note: We will use the same HTML call for all translation examples.

Style.css

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

Browser output:

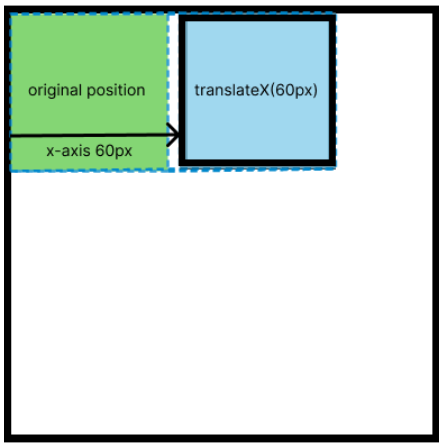


- Move the element 60 pixels to the right using translateX():

style.css:

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

Browser output with some required information for better understanding

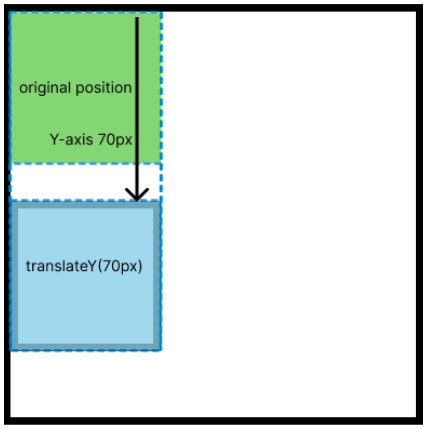


- Move the element 70 pixels to the bottom using translateY():

Style.css

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

Browser output with some required information for better understanding



b. Scale:

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

Syntex:

```
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.

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

Index.html

```
Unset
<!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 the scale() examples.

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

Style.css:

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

Browser output:

before	after
	

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

Style.css:

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

Browser output:

before	after
	

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

Style.css:

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

Browser output:

before	after
	

3. Rotate:

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

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

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

Here are some examples of how to use rotate() in CSS;

Index.html

```
Unset
<!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"></img>
</body>
</html>
```

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

1) Rotate an element 45 degrees clockwise:

Style.css:

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

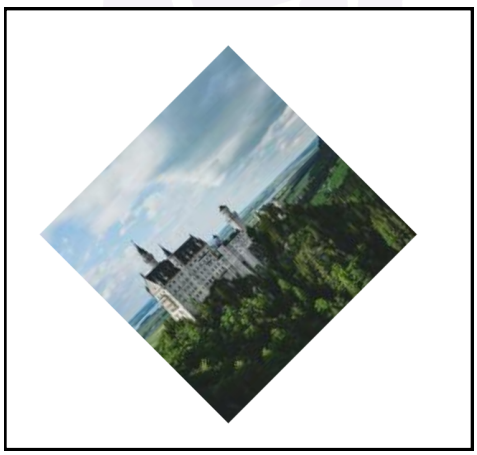
Browser output:

2) Rotate an element 45 degrees anticlockwise:

Style.css:

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

Browser output:



4. Skew

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

Syntex:

```
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.

Here are some examples of how to use skew() in CSS;

Index.html

```
Unset
<!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>
```

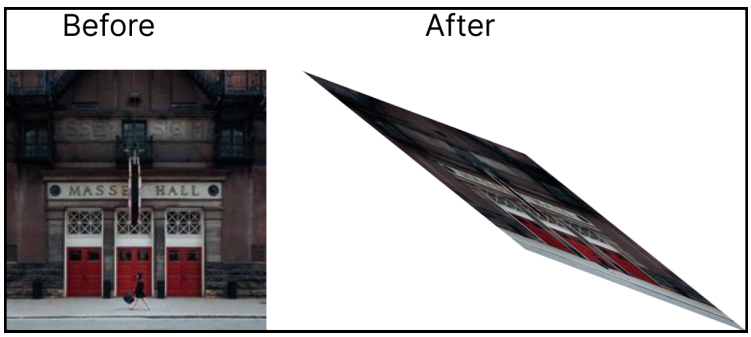
Note: We will use the same HTML code for all skew examples.

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

style.css:

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

Browser output:

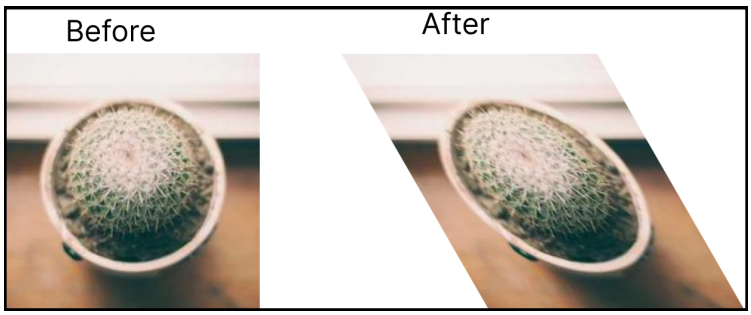


2) Skew an element 30 degrees in the X direction using **skewX():**

style.css

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

Browser output:



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

style.css

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

Browser 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:

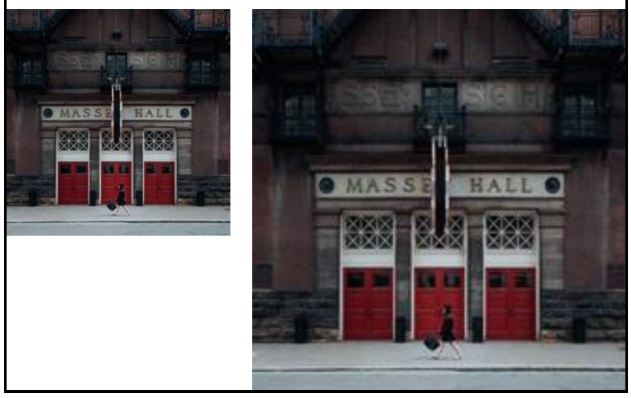
1) Scale the element in horizontal and vertical directions:

syntex:

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

Browser output:

Before After



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()

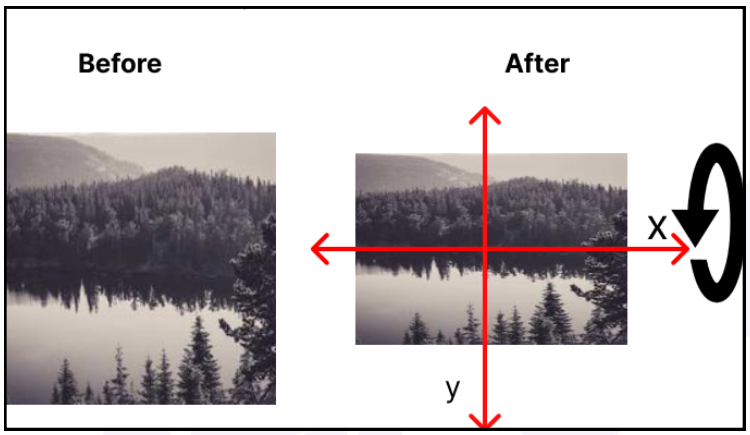
The rotateX() transformation in CSS allows you to rotate an element around the X-axis in 3D space. This rotation creates a tilting or flipping effect along the horizontal axis.

Rotate the element for its pivot in x direction.

style.css

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

Browser output



2. rotateY():

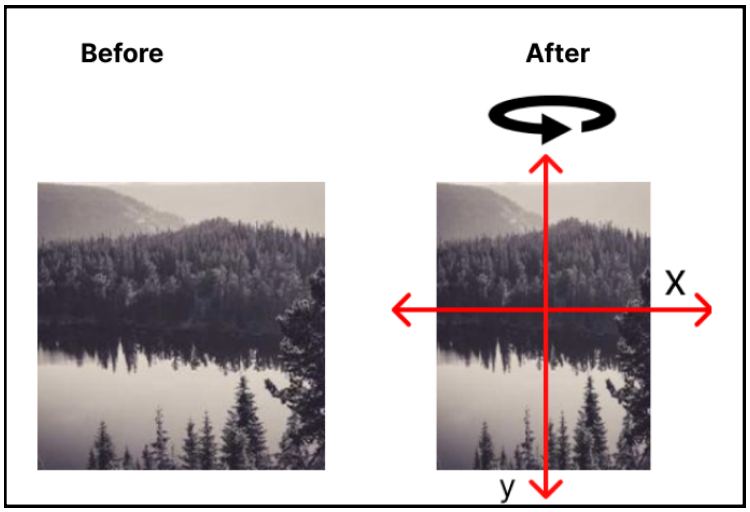
The rotateY() transformation in CSS allows you to rotate an element around the Y-axis in 3D space. This rotation creates a horizontal flipping or spinning effect.

Rotate the element for its pivot in y direction.

style.css

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

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);
}
```

3. rotateZ():

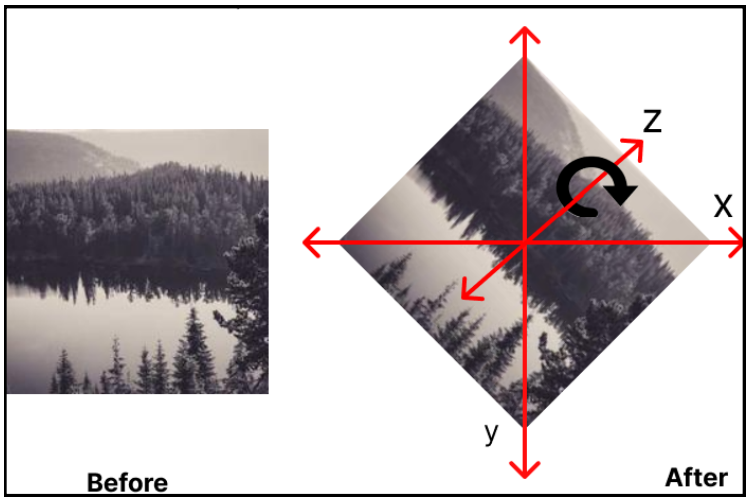
The rotateZ() transformation in CSS allows you to rotate an element around the Z-axis in 3D space. This rotation creates a twisting or spinning effect, similar to turning a wheel on its axis.

Rotate the element for its pivot in z direction.

style.css

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

Browser 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.