Transition and Transformation:

Transition:

- A transition is a way to **smoothly** change a property from one value to another over a specified duration. It provides a way to control the animation between two states.
- Transitions enable you to define the transition between two states of an element. Different states may be defined using pseudo-classes like :hover or :active or dynamically set using JavaScript.
- To create a transition effect, you must specify two things:
    1. the CSS property you want to add an effect to
    2. the duration of the effect

Note: If the duration part is not specified, the transition will have no effect, because the default value is 0.

Transition properties:

- transition-property
- transition-duration
- transition-delay
- transition-timing-function
- transition: it is the shortcut to apply above 4 properties

Syntax:

```
element {
```

```
    transition: property duration timing-function delay;
}
```

1. **property:** The CSS property you want to apply the transition to (e.g., color, width, opacity). The default of the transition-property is all. If we set its value to none then transition will not be applied to any properties.
2. **duration:** The time it takes for the transition to complete (in seconds or milliseconds).
3. **timing-function:** Specifies the speed curve of the transition (e.g., ease, linear, ease-in-out).
4. **delay:** Optional. Delays the start of the transition for a specified time.

Example1:

```
<!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>

    .box{
        width: 200px;
        height: 200px;
        background-color: green;
       transition-property: width;
       transition-duration: 2s;
    }

    .box:hover{
      width: 400px;
    }
```

```
  </style>


</head>
<body>

<div class="box"></div>


</body>
</html>
```

Example2: applying multiple properties

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Transition Example</title>
    <style>
        .box {
            width: 200px;
            height: 200px;
            background-color: green;

            /* Applying transition to width and height properties */
            transition-property: width, height;
            transition-duration: 2s, 2s;

            /* You can also apply transitions to more properties as
follows */
            /*
            transition-property: width, height, background-color;
```

```
                transition-duration: 2s, 4s, 5s;
                */
            }

            .box:hover {
                width: 400px;
                height: 400px;
                /* background-color: aqua; */
            }
        </style>
    </head>
    <body>
        <div class="box"></div>
    </body>
    </html>
```

Example3: Making rectangle to circle.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Transition Circle Example</title>
  <style>
    .box {
      width: 200px;
      height: 200px;
      background-color: red;

      /* This will transition only width and height,
         background-color and border-radius will change instantly without
transition */
      /*
      transition-property: width, height;
      transition-duration: 2s;
      */
```

```
      /* To transition all properties, use: */

      transition-property: width, height, background-color, border-radius;


      /* Or use all for a simpler approach */

      /* transition-property: all; */
      transition-duration: 2s;

    }

    .box:hover {
      width: 400px;
      height: 400px;
      background-color: blueviolet;
      border-radius: 50%;
    }
  </style>
</head>
<body>
  <div class="box"></div>
</body>
</html>
```

Important Notes on CSS Transitions:

1. **Using `transition-duration` alone:**

- When you specify only `transition-duration`, without `transition-property`, the default behavior is that all animatable properties will be transitioned. This means the browser will automatically apply the transition to any property that changes.

```
.box {
  width: 200px;
  height: 200px;
  background-color: red;
```

```
  transition-duration: 2s; /* This will transition all properties by
default */
}

.box:hover {
  width: 400px;
  height: 400px;
  background-color: aqua;
  border-radius: 50%;
}
```

This is equivalent to:

```
.box {
      transition: all 2s; /* Transitions all properties */
      }
```

2. **Using `transition` shorthand:**

- You can use the `transition` shorthand to define both the properties and the duration together. This shorthand makes it easier to write transitions.

  Example:

```
.box {
      transition: width 2s, height 2s;
      }
```

This is the same as writing:

```
.box {
       transition-property: width, height;
      transition-duration: 2s, 2s;
 }
```

3. **Multiple Transitions:**

- You can apply multiple transitions to different properties with different durations by separating them with commas, like so:

```
transition: width 2s, height 4s, background-color 5s;
```

4. **Using `transition: all`:**

- If you want all animatable properties to be transitioned, use `transition: all 2s;`. This means any property that changes (like `width`, `height`, `background-color`, etc.) will be smoothly animated.

Timing function:

- In CSS transitions and animations, the **timing-function** property is used to define the speed curve of the transition.
- It determines how the intermediate values between the starting and ending states of an element should be calculated over time.
- The timing function is applied to the transition or animation to control the acceleration and deceleration of the changes.

Syntax:

```
transition-timing-function: ease;
```

Here are some common timing functions:

1. **ease:** slow start, then fast, then end slow
    - This is the default timing function.
    - It starts slowly, accelerates in the middle, and slows down again towards the end.
    - It's often considered a good choice for most animations as it provides a natural and pleasant effect.

2. **linear:** same speed from start to end
    - Progresses at a constant speed from the beginning to the end.
    - There is no acceleration or deceleration.

3. **ease-in:** slow start
    - Starts slowly and accelerates quickly towards the end.

4. **ease-out:** slow end
    - Starts quickly and decelerates slowly towards the end.

5. **ease-in-out:** slow start and end
    - Combination of ease-in and ease-out.
    - It starts slowly, accelerates in the middle, and then decelerates towards the end.

6. **cubic-bezier(n, n, n, n):**
    - A custom cubic Bézier function where you can define your own acceleration curve using four values (n1, n2, n3, n4).
    - Each value must be between 0 and 1.

      Example:

      transition-timing-function: cubic-bezier(0.91,-0.16, 0.58, 1);

Tips to generate the cubic-bezier:

- Specify the transition-timing-function: linear;
- Open the developer tool, and inspect this property.
- There will be a small box beside this property, by stretching the curve we can generate the desired cubic-bezier.

7. **steps:**
    - It performs the animation in steps

      Example:

      transition-timing-function: steps(4)

Example3:

```
<!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>

      div {
         height: 200px;
         width: 200px;
         background-color: pink;
         transition-property: background-color;
         transition-duration: .5s;
         transition-timing-function:  ease-in-out;
         /* transition-timing-function:  steps(4); */

         /* Shorthand for above 3 properties */
         /* transition: background-color 3s ease-in-out; */


      }


      div:hover {
          background-color: #3498db;
      }


/* If we keep pressing mouse button then this div will be in active state
*/
      /* div:active {
          background-color: #3498db;
      } */
```

```
    </style>


</head>
<body>


<div class="box"></div>

</body>
</html>
```

In this example, the background color of the div will smoothly transition over the 3 seconds with an ease-in-out/steps timing function when hovered.

Example4:

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


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        .box {
            width: 200px;
            height: 200px;
            background-color: coral;
            transition: width 3s ease-in-out, background-color 3s steps(4);

        }


        .box:hover {
```

```
        width: 400px;
        background-color: #47ff5c;
      }
    </style>
</head>


<body>
    <div class="box"></div>
</body>


</html>
```

Lab Assignment:

- **Create an input field with initial width 200px and when we focus on this input field the width of the input field should be transited to 300px.**

Solution:

```
<!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>
      input {
        width: 200px;
        transition: width 2s ease;
      }

      input:focus {
        width: 300px;
      }
    </style>

</head>

<body>


    <input type="text">


</body>

</html>
```

Student task:

Create a navigation menu with items like AboutUs, ContactUs, Home and when we hover over them, show the transition effect on their background-color and text color.

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

```
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <style>


     ul{
       display: flex;
       gap: 20px;

     }

     ul>li{
       list-style: none;

     }

     a{
       font-size: 1.5rem;
       text-decoration: none;
       padding: 10px;
       transition: color 0.3s ease, background-color 0.3s ease, border 0.3s ease;
     }



     a:hover{
       color: white;
       background-color: darkcyan;
       border: 2px solid black;

     }

</style>
</head>

<body>
```

```
    <ul>
      <li><a href="#">AboutUs</a></li>
      <li><a href="#">ContactUs</a></li>
      <li><a href="#">Home</a></li>
    </ul>

</body>

</html>
```

Transform 2D(positioning and angle):

- **CSS transforms** allow you to modify the appearance of an element in **2D or 3D space**. Common transformations include **scaling**, **rotating**, **skewing**, and **translating** elements. This powerful tool is essential for creating dynamic and interactive layouts and effects.

Syntax:

```
element {
        transform: transform-function;
    }
```

Types of Transformations:

- CSS transforms enable various transformations to HTML elements, including:

1. Translation (Moving):

- It allows for flexible positioning without altering the document flow.
    - `translate(x, y)`: Moves an element from its current position based on the specified values for the X-axis and Y-axis.
    - `translateX(x)`: Moves the element along the X-axis.
    - `translateY(y)`: Moves the element along the Y-axis.
    - `translate(100%)`: This shorthand function is equivalent to `translate(100%, 0%)`.meaning:
        - X-axis: Move the element to the right by 100% of its own width.

- Y-axis: No vertical movement (0%).

Note: The values for `x` and `y` can be specified in different units:

- Pixels (px): Fixed-size movement.
- Percentages (%): Moves relative to the element's own dimensions. For example, `translate(50%, 50%)` mov-es the element halfway of its width and height.

Negative Values: You can use negative values to move elements in the opposite direction.

Example:

- `translateY(-50px):` moves the element 50 pixels upward.

2. Rotation:

- **`rotate(angle)`**: Rotates the element clockwise around a specified point, usually the center.

  Example:

  ```
  transform: rotate(45deg); /* Rotates the element 45 degrees clockwise */
  ```

  Negative value: it rotates anti clockwise.

  ```
  transform: rotate(-45deg); /* Rotates the element 45 degrees anti-clockwise */
  ```

3. Scaling (Resizing):

- **`scale(x, y)`**: Increases or decreases the size of an element based on the specified values for width (X-axis) and height (Y-axis).
- **`scaleX(x)`**: Scales the element along the X-axis.
- **`scaleY(y)`**: Scales the element along the Y-axis.

4. Skewing (Distorting):

- **`skew(x-angle, y-angle)`**: Distorts the element along the X and Y axes by the specified angles.
- **`skewX(angle)`**: Skews the element horizontally.
- **`skewY(angle)`**: Skews the element vertically.

Important Note

- Only transformable elements can be transformed. Most HTML elements are transformable whereas common non-transformable properties/elements include:
    - `<html>`, `<head>`, `<title>`, `<meta>`, `<base>`, `<link>`, `<style>`, `<script>`, `<noscript>`, `<iframe>`, `<form>`, `<body>`, `<template>`.

Example:

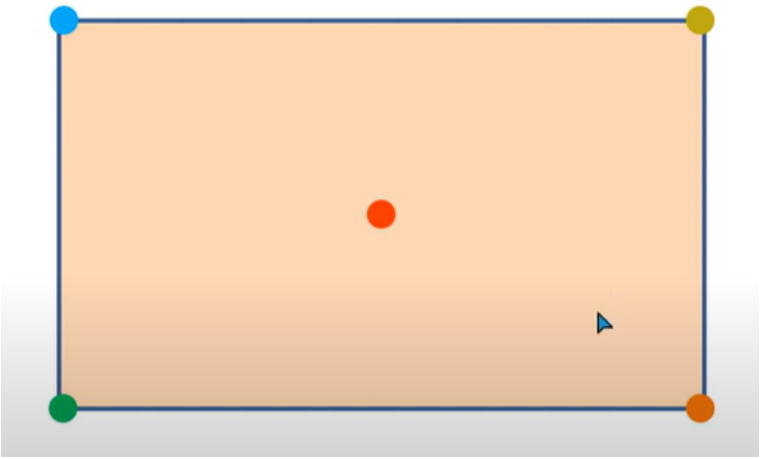
```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>



        .box {
            width: 200px;
            height: 100px;
            background-color: #3498db;
            margin: 150px 300px;
            /* transform: rotate(45deg);  rotate clockwise it is
equivalent to rotateZ(45deg)*/
           /* transform: rotate(-45deg); */
```

```
            /* rotateX and rotateY will show effect with 3d perspective*/

             /* transform: rotateX(45deg); */
            /* transform: rotateY(45deg); */



            /* transform: translate(200px, 400px); */
            /* transform: translateX(-300px); */

            /* element will be moved horizontally (along the X-axis) by
100% of its own width. */
            /* for example, 200 pixels wide, translateX(100%) will move it
200 pixels to the right from its original position. */
            /* transform: translateX(100%); */

            /* Scale x and Y by 1.5 times  */
            /* transform: scale(1.5);  */

            /* Scale x and y both by 200% means x become 400px and y
become 200px */
            /* transform: scale(200%);  */

            /* Scale x 3 time and y 2 times*/
            /* transform: scale(2,3); */

            /* the element will be tilted horizontally 45deg. it is
equivallent for skew(10deg, 0deg) */
            /* Imagine you have a rectangle, and you push one side to the
left or right while keeping the other side fixed. This is essentially what
skewing does! */
            /* transform: skew(10deg); */

            /* transform: skewX(45deg); */
          /* When skewX(90deg) then would turn the element into a vertical
line. not visible */
          /* Skew in vertical direction */
          transform: skewY(45deg);
```

```
            /* transform: skew(30deg, 20deg); */


              /* transform: rotate(45deg) scale(1.2); */
              /* transform: translate(20px, 40px) rotate(45deg) scale(1.5);
*/
          }
    </style>
</head>


<body>
    <div class="box">Welcome</div>
</body>


</html>
```

Transform-origin:

- The `transform-origin` property in CSS allows you to specify the point around which a transformation is applied.
- The default transform-origin in CSS is the **center of the element**.



Syntax:

```
transform-origin: x-axis, y-axis;
```

x-axis possible values:

- left
- right
- center
- %

y-axis possible values:

- top
- bottom
- center
- %

Example:

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


<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <style>
      .box {
         width: 200px;
         height: 100px;
         background-color: #3498db;
         margin: 150px 300px;
         transform: rotate(30deg);
```

```
            transform-origin: left bottom;

            /* 0% 0% means the top-left corner */
            /* transform-origin: 0% 0%;  */

            /* 100% 100% means the bottom-right corner */
            /* transform-origin: 100% 100%;  */




        }
    </style>
</head>


<body>
    <div class="box">Welcome</div>
</body>


</html>
```

Note: CSS transform provides a flexible way to enhance the visual appearance of web content without modifying the underlying HTML structure. It enables developers to create engaging animations, design effects, and responsive layouts with ease. By using transforms, you can achieve complex visual effects that were previously only possible with JavaScript or Flash.

Transition + Transformation:

- CSS transitions can be applied to transforms to create smooth animations.

Example1:

```
<!DOCTYPE html>
```

```html
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>

    <style>
        .box {
            width: 100px;
            height: 100px;
            background-color: #3498db;
            transition: transform 2s ease-in-out;
        }

        .box:hover {
            transform: rotate(90deg);
        }
    </style>

</head>

<body>

    <div class="box"></div>

</body>

</html>
```

Example2:

```
<!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>
      .box {
         width: 200px;
         height: 150px;
         background-color: green;
         transition: transform 2s ease-in-out, background-color 0.3s
ease-in-out;
      }

      .box:hover {
         transform: scale(1.5);
       /* transform: scale(1.5) rotate(90deg); */
         background-color: red;

      }
   </style>



</head>

<body>
```

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


</body>

</html>
```

Example3:

```
<!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>
        .btn {
            padding: 10px;
            background-color: #3498db;
            color: white;
            transition: transform 0.3s ease-in-out;
        }

        .btn:active {
            transform: rotate(360deg) scale(1.2);
        }
    </style>



</head>
```

```
    <body>

      <button class="btn">Click</button>


    </body>

    </html>
```

Example4:

```
<!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>
        .image {
            width: 200px;
            height: 150px;
            transition: transform 2s ease-in-out;
        }


        .image:hover {


            transform: translateX(400px);
            /* transform: translateX(400px) translateY(300px); */

        }
    </style>
```

```
</head>

<body>

    <img
src="https://www.wildnatureimages.com/images/640/070620-014-The-Tetons.jpg
" class="image" width="200px" height="200px">

</body>

</html>
```

Example5:

```
<!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>
      .box {
         width: 150px;
         height: 100px;
         background-color: #2ecc71;
```

```
        transition: transform 0.3s ease-in-out, opacity 0.3s ease-in-out;
      }

      .box:hover {
        transform: skew(20deg);
        opacity: 0.5;
      }
    </style>



</head>

<body>

  <div class="box"></div>

</body>

</html>
```

Student Task: Centered Image with Hover Effect

Objective: Center an image on the page and apply a hover effect to scale it up.

Instructions:

1. Insert an Image: Add an image of your choice with a width of 300px.
2. Center the Image: Use CSS to position the image in the center of the page both vertically and horizontally.
3. Apply Hover Effect: Implement a hover effect that scales the image up by a factor of 1.5.
4. Add Transition: Ensure the scaling effect is smooth by adding a transition.

Solution:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Hover Zoom</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;

        }

        .image {
            width: 300px; /* Image width */
            transition: transform 0.5s ease; /* Smooth transition */
        }

        .image:hover {
            transform: scale(1.5); /* Zoom in on hover */
        }
    </style>
</head>
<body>
    <img class="image"
src="https://www.wildnatureimages.com/images/640/070620-014-The-Tetons.jpg
" alt="Sample Image">
</body>
</html>
```

Animations:

- CSS animations allow for more complex dynamic movements compared to transitions. They enable the creation of multi-step keyframe-based animations that can change an element's properties (like size, position, color,

and opacity) over a specified duration, resulting in smooth and visually appealing effects.

Key Components of CSS Animations

1. **Keyframes**: Define the styles at various points in the animation timeline.
2. **Animation Properties**: Control how the animation behaves.

Syntax:

```
@keyframes animation-name {
from {
   /* styles at the start of the animation */
}

to {
   /* styles at the end of the animation */
}
}

element {

      animation: animation-name duration timing-function delay
      iteration-count direction fill-mode;

}
```

- **animation-name:** The name of the keyframes rule.
- **duration**: The time it takes for the animation to complete (in seconds or milliseconds).
- **timing-function**: Specifies the speed curve of the animation (e.g., ease, linear, ease-in-out).
- **delay**: Optional. Delays the start of the animation for a specified time.
- **iteration-count**: Specifies the number of times the animation should repeat (e.g., `infinite` for continuous animation or we can specify any number to be repeated).

- **direction**:Specifies whether the animation should play in reverse (e.g., `reverse`, `alternate`).
- **fill-mode**: Specifies what values are applied by the animation outside the time it is executing (e.g., `forwards`, `backwards`).
    - The `forwards` value ensures that the element retains the styles from the last keyframe (in this case, the 100% state) **after** the animation completes.
    - Without `forwards`, the element would revert back to its original state once the animation finishes.
    - With `forwards`, the element holds on to the final state (e.g., red background color at the 100% keyframe).

Example1 : Simple Slide-in Animation

```
<!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>
      @keyframes slide-in {
         from {
            transform: translateX(-100%);
         }

         to {
            transform: translateX(0);
         }
      }

      .cl1 {
         background-color: aqua;
```

```
        animation: slide-in 1s ease-out;
    }
  </style>



</head>

<body>

  <h1 class="cl1">Welcome to Chitkara</h1>

</body>

</html>
```

CSS animations offer many benefits. They make websites more engaging and interactive, helping to grab users' attention and guide them through the content. Additionally, animations can provide feedback to users, improving usability and overall user experience. Moreover, they can add a touch of professionalism and creativity to a website's design without relying on heavy JavaScript or Flash.

Example2: Multi-Step Slide-in Animation

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    @keyframes slide-in {
      0% {
        transform: translate(-100%);
```

```
        }

        50% {
            transform: translateX(50%);
            opacity: 0.5;
        }

        100% {
            transform: translateX(0);
            opacity: 1;
        }
    }

    .box {
        width: 200px;
        height: 100px;
        background-color: #3498db;
        animation: slide-in 2s ease-out;
    }
  </style>
</head>

<body>
  <div class="box"></div>
</body>

</html>
```

In the above example, the .box element will:

- Start with translateX(-100%) and opacity: 1 at 0%.
- Move to translateX(50%) and opacity: 0.5 at 50%.
- Finally, end with translateX(0) and opacity: 1 at 100%.
- The 50% keyframe introduces a midpoint state during the animation. You can use as many percentage points as needed to create the desired effect.

Example1: scale element

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

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <style>
      @keyframes scale {
         0% {
            transform: scale(1);
         }

         50% {
            transform: scale(1.2);
         }

         100% {
            transform: scale(1);
         }
      }

      .box {
         width: 100px;
         height: 100px;
         background-color: coral;
         animation: scale 2s infinite ease-in-out;

       /* Here iteration count will be 2 times only */
       /* animation: scale 2s 2 ease-in-out; */

      }
   </style>
</head>
```

```
<body>
  <div class="box"></div>
</body>

</html>
```

Example2: Fade-in text

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <style>
    @keyframes fadeIn {
      0% {
        opacity: 0;
      }

      100% {
        opacity: 1;
      }
    }

    .cl1 {
      font-size: 24px;
      color: navy;
      animation: fadeIn 2s infinite ease-in-out;

    }
  </style>
</head>
```

```
    <body>
      <p class="cl1">Welcome to the Example Page!</p>
    </body>

    </html>
```

Example3: rotating image

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


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        @keyframes rotate {
            0% {
                transform: rotate(0deg);
            }


            100% {
                transform: rotate(360deg);
            }
        }


        .rotate-image {
            width: 200px;
            height: 200px;
            background-image:
url(https://www.wildnatureimages.com/images/640/070620-014-The-Tetons.jpg)
;
            animation: rotate 3s infinite linear;
            background-position: center;
            background-size: cover;
```

```
        }
    </style>
</head>


<body>
    <div class="rotate-image"></div>
</body>


</html>
```

Example4 : bouncing ball

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1.0">
 <style>
  @keyframes bounce {
   0%, 20%, 50%, 80%, 100% {
    transform: translateY(0);
   }

   /* 25%{
        transform: translate(20px,30px);
     } */

   40% {
    transform: translateY(-30px);
   }
   60% {
```

```
      transform: translateY(-15px);
    }
  }

  .bounce-ball {
    width: 50px;
    height: 50px;
    background-color: green;
    border-radius: 50%;
    animation: bounce 2s infinite ease-in-out;
  }
 </style>
</head>
<body>
 <div class="bounce-ball"></div>
</body>
</html>
```

Example5 : Color Cycle text:

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <style>
    @keyframes colorCycle {
      0% {
        color: red;
      }

      25% {
        color: blue;
      }
```

```
            50% {
                color: yellow;
            }

            75% {
                color: green;
            }

            100% {
                color: red;
            }
        }

        .color-cycle-text {
            font-size: 24px;
            font-weight: bold;
            animation: colorCycle 4s infinite linear;
        }
    </style>
</head>

<body>
    <p class="color-cycle-text">CSS Color Cycling</p>
</body>

</html>
```

Example 6: Pulse circle

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

<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <style>
        @keyframes pulse {
            0% {
                transform: scale(1);
            }

            50% {
                transform: scale(1.2);
            }

            100% {
                transform: scale(1);
            }
        }

        .pulse-circle {
            width: 100px;
            height: 100px;
            background-color: purple;
            border-radius: 50%;
            animation: pulse 3s infinite ease-in-out;
        }
    </style>
</head>

<body>
    <div class="pulse-circle"></div>
</body>

</html>
```

Example7: Shaking heading

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

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <style>
      @keyframes shake {

         0%, 100% {
            transform: translateX(0);
         }

         25%, 75% {
            transform: translateX(-10px);
         }

         50% {
            transform: translateX(10px);
         }
      }

      .shake-heading {
         font-size: 36px;
         font-weight: bold;
         animation: shake 2s infinite ease-in-out;
      }
   </style>
</head>

<body>
   <h1 class="shake-heading">Shaking Heading</h1>
</body>

</html>
```

Example 8: Fading background

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <style>
    @keyframes fadeInBackground {
      0% {
        background-color: yellow;
      }
      50% {
        background-color: aqua;
      }
      100% {
        background-color: yellow;
      }
    }

    body {
      animation: fadeInBackground 4s infinite linear;
    }
  </style>
</head>
<body>
  <h2>This is a fading background example</h2>
</body>
</html>
```

Example9: bar chart:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <style>
        @keyframes growBar {
            0% {
                height: 20px;
            }

            50% {
                height: 150px;
            }

            100% {
                height: 20px;
            }
        }

        .bar-chart {
            width: 50px;
            background-color: #3498db;
            animation: growBar 3s infinite ease-in-out;
        }
    </style>
</head>

<body>
    <div class="bar-chart"></div>
</body>

</html>
```

Example10: Spinning Loader

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Spinning Loader</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f2f2f2; /* Background color */
        }

        .loader {
            border: 8px solid #3498db; /* Outer color */
            border-top: 8px solid transparent; /* Transparent top */
            border-radius: 50%; /* Circle shape */
            width: 60px;
            height: 60px;
            animation: spin 1s linear infinite; /* Animation definition */
        }

        @keyframes spin {
            0% {
                transform: rotate(0deg); /* Initial rotation */
            }
            100% {
                transform: rotate(360deg); /* Full rotation */
            }
        }
    </style>
</head>
<body>
    <div class="loader"></div>
</body>
</html>
```

CSS Scroll Based Animation:

- CSS scroll-driven animations allow us to animate elements based on the user's scrolling behavior, rather than over time like traditional animations. With scroll-based animations, the position of elements is animated as the user scrolls down or up a webpage, providing a dynamic and engaging effect without needing JavaScript.

Types of Scroll-Based Timelines:

- Scroll-based timelines allow CSS animations to progress based on the user's scroll position rather than being time-based. The animation updates as the page is scrolled up or down (or left or right).

There are two types of scroll-based animations:

1. **Scroll Progress Timeline**
    - As you scroll from top to bottom (or left to right), the element's position is updated. The animation progresses from 0% (when you start scrolling) to 100% (when you finish scrolling).
2. **View Progress Timeline**
    - This is based on how much of an element is visible in the viewport (the visible part of the webpage). The animation progresses as the element comes into and out of view.

Key Properties for Scroll-Based Animations:

1. `animation-timeline`:
    - This property specifies which timeline will control the animation progress. It can be set to scroll-based or view-based timelines instead of the default time-based document timeline.

    Values:

- **`scroll()`**: Ties the animation progress to the scroll progress of an element (scroll progress timeline).
- **`view()`**: Ties the animation progress to the visibility of an element within a scroller (view progress timeline).

Example:

```
    animation-timeline: scroll(); /* Animation based on scrolling */
    animation-timeline: view(); /* Animation based on element
visibility */
```

2. **`animation-range`**:
    - This property defines the range in which the animation will take place, based on the scroll or view progress timeline. It specifies the start and end points for the animation on the timeline.

    Values:

    - **`start-end`**: A range specified in terms of scroll position (e.g., pixels or viewport dimensions).

      Example:

      ```
        animation-range: 0 100vh; /* Animation starts at 0px and
      ends at 100vh of scrolling */
      ```

Example: Changing the background color based on the scroll.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scroll Background Color Change</title>
    <style>
        body {
            height: 200vh; /* Set the page height to make it scrollable */
```

```
            margin: 0;
            background-color: pink; /* Initial background color */
            animation: bg-color-change forwards;
            animation-timeline: scroll(); /* Use scroll-based timeline */
            animation-range: 0 100vh; /* Animate based on scroll from 0 to
100% */
        }


        @keyframes bg-color-change {
            0% {
                background-color: pink; /* Initial color at 0% scroll */
            }
            50% {
                background-color: yellow; /* Color changes to yellow at
50% scroll */
            }
            100% {
                background-color: red; /* Color changes to red at 100%
scroll */
            }
        }
    </style>
</head>
<body>
</body>
</html>
```

Example: Text Color Change on Scroll

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scroll Text Color Change</title>
    <style>
        body {
            height: 200vh;
            margin: 0;
```

```
            display: flex;
            justify-content: center;
            align-items: center;
            font-size: 3rem;
            animation-timeline: scroll();
            animation-range: 0 100vh;
        }

        h1 {
            animation: text-color-change forwards;
            animation-timeline: scroll();
            animation-range: 0 100vh;
        }

        @keyframes text-color-change {
            0% {
                color: red;
            }
            50% {
                color: blue;
            }
            100% {
                color: green;
            }
        }
    </style>
</head>
<body>
    <h1>Scroll to Change Text Color</h1>
</body>
</html>
```

Example: Rotate Animation on Scroll

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

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scroll Rotate Animation</title>
    <style>
        body {
            height: 200vh;
            background-color: lightyellow;
            display: flex;
            justify-content: center;
            align-items: center;
            font-size: 3rem;

        }


        h1 {
            animation: rotate-change forwards;
            /* animation-timeline: view(); */
            animation-timeline: scroll();
            animation-range: 0 100vh;
        }


        @keyframes rotate-change {
            0% {
                transform: rotate(0deg);
            }

            50% {
                transform: rotate(180deg);
            }

            100% {
                transform: rotate(360deg);
            }
        }
    </style>
</head>
```

```
<body>
    <h1>Scroll to Rotate Text</h1>
</body>

</html>
```

Example: Sliding in-out divs:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simplified Scroll Animation</title>
    <style>
        /* Page scroll setup */
        body {
            height: 250vh; /* Allow enough height to make the page
scrollable */

            background-color: gray;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        /* Keyframes for sliding in animation */
        @keyframes slide-in {
            100% {
                transform: translateX(0); /* Move the element to its
original position */
            }
        }

        /* Container for the boxes */
        #box {
            width: 50%; /* Set container width to 50% of the viewport */
            height: 50vh; /* Container height is half of the viewport
height */
```

```
            display: grid;
            grid-template-columns: repeat(2, 1fr); /* Two-column grid
layout */

            gap: 10px; /* Space between the boxes */
        }

        /* Styling for individual boxes */
        #box div {
            background-color: yellowgreen; /* Set box background color */
            height: 100px; /* Set a fixed height for the boxes */
            animation: slide-in forwards; /* Apply slide-in animation */
            animation-timeline: view(); /*  Animation triggered by view */
            animation-range: 0 100vh; /*  Define the animation range */
        }

        /* Odd-numbered boxes slide in from the left */
        #box div:nth-of-type(odd) {
            transform: translateX(-50vw); /* Start off-screen from the
left */
        }

        /* Even-numbered boxes slide in from the right */
        #box div:nth-of-type(even) {
            transform: translateX(50vw); /* Start off-screen from the
right */
        }
    </style>
</head>
<body>

    <!-- Container with sliding boxes -->
    <div id="box">
        <div></div>
        <div></div>
        <div></div>
        <div></div>
        <div></div>
        <div></div>
    </div>
```

```
</body>
</html>
```

Here Initially odd divs are off-screen from the left side and even number divs are off-screen from the right side, and when scroll animation occurs (keyframe becomes 100%)they will come to the actual position.

CSS Transform 3D:

- CSS 3D transforms allow you to manipulate elements in a three-dimensional space.

3D transform values:

- **rotate(*angle*):** rotate the element clockwise. Similar to rotateZ.
- **rotateX(angle):** Positive angles (e.g., rotateX(45deg)) tilt the element upwards. Whereas Negative angles (e.g., rotateX(-45deg)) tilt the element downwards.
- **rotateY(angle):** Positive angles (e.g., rotateY(45deg)) tilt the element to the right. Whereas Negative angles (e.g., rotateY(-45deg)) tilt the element to the left.
- **rotateZ(angle):** rotateZ(45deg)) rotate the element clockwise. It is similar to the rotate() function only but it is used in 3D space with the combination of other functions.
- **rotate3d(x,y,z, angle):** Defines a 3D rotation

- **translate(*x,y*)**
- **translateX(x)**
- **translateY(y)**
- **translateZ(z):** Defines a 3D translation, using only the value for the Z-axis
- **translate3d(x,y,z):** Defines a 3D translation

- **scale(x,y)**
- **scaleX(x)**
- **scaleY(y)**

- **scaleZ(z):** Defines a 3D scale transformation by giving a value for the Z-axis
- **scale3d(x,y,z):** Defines a 3D scale transformation

- **perspective(n):** Defines a perspective view for a 3D transformed element

Important Note:

- In order to perform CSS transformation in 3d space, we have to start by configuring the 3D space by giving it a perspective.
- `perspective` only affects **3D transformations**, specifically those that involve depth, like `translateZ()`, `rotateX()`, `rotateY()`, or `scaleZ()`.

Perspective:

- The first element to set is the perspective, The perspective is what gives us the 3D impression. The farther from the viewer the elements are, the smaller they are.
- The perspective property is used to give a sense of depth to 3D transformed elements.

  https://3dtransforms.desandro.com/perspective

- Perspective can be applied in two ways:

1. With the transform property, with perspective() as a function:
   Example: inside the child element along with the other transform property.

```
transform: perspective(400px) rotateY(45deg);
```

2. With the perspective property:
   Example: inside the parent element. And in the child element we can use the other transform property.

```
perspective: 400px;
```

Example:

```
<!DOCTYPE html>
```

```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{
            width: 300px;
            height: 300px;
            background-color: gray;
            /* perspective: 400px; */
        }

        .box{
            width: 300px;
            height: 300px;
            background-color: aquamarine;

            /* transform: rotateX(45deg); */

            transform: perspective(400px) rotateX(45deg);

        }

    </style>

</head>
<body>


    <div class="container">
        <div class="box"></div>
    </div>

</body>
</html>
```

Example: Using transition:

```
<!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{
            width: 300px;
            height: 300px;
            background-color: gray;

        }


        .box{
            width: 300px;
            height: 300px;
            background-color: aquamarine;

            transition: transform 2s ease;
        }

        .box:hover{
            transform: perspective(400px) rotateX(45deg);
        }

    </style>


</head>
<body>
```

```
    <div class="container">
        <div class="box"></div>
    </div>


</body>
</html>
```

Note: As we know the default transform-origin is center, but if we change the origin point then the child div will rotate from that point in the X direction.(horizontal).

Example

```
        .container {
            width: 300px;
            height: 300px;
            background-color: gray;
        }

        .box {
            width: 300px;
            height: 300px;
            background-color: red;
            transform-origin: bottom;
           /* transform-origin: right bottom; */
            transform: perspective(400px) rotateX(60deg);

           /* transform-origin: top;
            transform: perspective(500px) rotateX(-60deg); */


        }
```

To rotate in Y axis:

```
transform: perspective(500px) rotateY(45deg);
/* transform-origin: left; */
```

Or

```
transform: perspective(500px) rotateY(-45deg);
/* transform-origin: right; */
```

To rotate in Z axis:

```
transform: rotateZ(60deg);
```

Note:

- **Rotation Around the Z-Axis:** When you use rotateZ(), the rotation occurs in the plane of the element. Since the Z-axis is perpendicular to the screen, rotateZ() doesn't create depth; it only spins the element around itself.
- If you apply a perspective but only rotate around the Z-axis, the visual effect may not be dramatic because the rotation is flat, and depth perception isn't strongly influenced.
- To notice the effect of rotateZ() with perspective, you may need to combine it with other transformations, like rotateX() or rotateY(), which create more three-dimensional effects.

Example:

```
transform: perspective(400px) rotateZ(45deg) rotateX(45deg);
```

rotate3d:

- The rotate3d() function allows you to rotate an element in 3D space around an arbitrary axis defined by a vector. This function provides more control over the rotation compared to rotateX(), rotateY(), and rotateZ().

  Syntax:

```
transform: rotate3d(x, y, z, angle);
```

- x, y, z: These three values define the axis of rotation. They specify the direction of the rotation vector.
- For example, (1, 0, 0) means rotate around the X-axis.
- (0, 1, 0) means rotate around the Y-axis.
- (0, 0, 1) means rotate around the Z-axis.
- angle: This is the angle of rotation, specified in degrees (e.g., 45deg)

Example:

```
transform: perspective(400px) rotate3d(1, 1, 0, 45deg); /* Rotate 45 degrees around the vector (1, 1, 0) */
```

Example using transition:

```
<!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 {
            width: 300px;
            height: 300px;
            background-color: gray;

        }


        .box {
            width: 300px;
            height: 300px;
```

```
            background-color: aquamarine;

            transition: transform 2s ease;
        }

        .box:hover {
            transform: perspective(400px) rotate3d(1, 1, 0, 90deg);

        }
    </style>


</head>

<body>

    <div class="container">
        <div class="box"></div>
    </div>


</body>

</html>
```

Translate using perspective:

- **translateX() and translateY(): Move elements left, right, up, or down in a 2D plane. perspective does not affect them.**
- **translateZ(): Moves elements forward or backward in 3D space, and perspective helps create the illusion of depth for this transformation.**

Example:

```
<!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 {
            width: 300px;
            height: 300px;
            background-color: gray;

        }

        .box {
            width: 300px;
            height: 300px;
            background-color: yellowgreen;
            transform: perspective(400px) translateZ(-200px) ;

        }


    </style>

</head>

<body>


    <div class="container">
        <div class="box"></div>
    </div>

</body>

</html>
```

Student Task: apply the transition effect on the .box to show the effect of moving the .box closer and farther

Solution:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TranslateZ Closer and Farther</title>

    <style>
        .container {
            width: 300px;
            height: 300px;
            background-color: gray;
            perspective: 400px; /* Set perspective */
        }

        .box {
            width: 300px;
            height: 300px;
            background-color: yellowgreen;
            transform: translateZ(100px); /* Move closer to the viewer */
            transition: transform 2s ease;
        }

        .box:hover  {
            transform: translateZ(-200px); /* Move farther from the viewer
on hover */
        }
    </style>

</head>

<body>
    <div class="container">
        <div class="box"></div>
    </div>
</body>
```

```
</html>
```

Example: Combining translateZ along with other properties:

```
        .box:hover  {

      /* transform: perspective(600px) translateX(150px)
translateZ(150px)*/
      /* transform: perspective(600px) rotateY(90deg) translateZ(150px);
*/
      transform: perspective(600px) rotateY(-90deg) translateZ(150px);


        }
```

translate3d:

- The `translate3d()` function in CSS is used to move elements in 3D space by specifying values along the X, Y, and Z axes. It's part of CSS Transforms, allowing developers to apply 3D transformations to HTML elements, enhancing the perception of depth.

  Syntax:

  ```
  transform: translate3d(x, y, z);
  ```

  x: The translation along the X-axis (horizontal movement). Can be in pixels (px), percentage (%), or other units.
  y: The translation along the Y-axis (vertical movement). Can also be in pixels, percentage, etc.
  z: The translation along the Z-axis (depth, i.e., moving towards or away from the viewer). This determines how far the element appears to be from the viewer.

  Example:

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

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TranslateZ Closer and Farther</title>


    <style>
        .container {
            width: 300px;
            height: 300px;
            background-color: gray;

        }


        .box {
            width: 300px;
            height: 300px;
            background-color: yellowgreen;
            transition: transform 2s ease;
        }


        .box:hover  {
          /* Without perspective it will not show any effect in translateZ
*/
          /* transform: translate3d(50px, 50px, 100px); */
          transform: perspective(600px) translate3d(50px, 100px, 150px);

      /* transform: perspective(500px) translate3d(100px, 0, 150px); */

        }
    </style>


</head>


<body>
```

```
    <div class="container">
        <div class="box"></div>
    </div>
</body>


</html>
```

Example: Combining translate3D with other properties:

```
   transform: perspective(500px) rotateY(60deg) translate3d(100px, 100px,
100px);
```

ScaleZ using Perspective:

- The `scaleZ()` function in CSS is part of the 3D transformation functions and is used to scale an element along the Z-axis (depth axis) in 3D space. Unlike `scaleX()` and `scaleY()`, which affect the width and height, respectively, `scaleZ()` changes how "deep" an element appears, making it look larger or smaller based on its depth.

  Syntax:

  ```
  transform: scaleZ(scaleFactor);
  ```

  scaleFactor: A number representing the scaling factor along the Z-axis.

  - A value of `1` means no scaling (the default size).
  - A value greater than `1` increases the size (closer to the viewer).
  - A value less than `1` decreases the size (further from the viewer).

  Important Points:

1. **Z-axis scaling is part of 3D transforms**, so it needs to be combined with the `perspective` property or other 3D transformations for it to have a noticeable effect.
2. `scaleZ()` works in 3D space, so the effect might not be visible unless the element is rotated or translated along other axes (X or Y).
3. We need to apply scaleZ property **before** rotating or translating to visualise the effect.

Example1:

```
transform: perspective(500px) scaleZ(2) rotateY(-30deg);
```

Example2: ScaleZ with translate

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>TranslateZ Closer and Farther</title>

  <style>
      .container {
          width: 300px;
          height: 300px;
          background-color: gray;

      }

      .box {
          width: 300px;
          height: 300px;
          background-color: yellowgreen;
          transform: perspective(400px) translateZ(-100px);
          transition: transform 2s ease;
      }

      .box:hover  {
```

```
            transform: perspective(400px) scaleZ(2) translateZ(-100px);


        }
    </style>

</head>

<body>
    <div class="container">
        <div class="box"></div>
    </div>
</body>




</html>
```

scale3d:

- The scale3d() function in CSS is used to scale an element in three-dimensional space along the X, Y, and Z axes simultaneously. It allows you to resize an element while taking into account its depth, giving it a more immersive 3D effect.

  Syntax:

  ```
  transform: scale3d(scaleX, scaleY, scaleZ);
  ```

  scaleX: A number representing the scaling factor along the X-axis (width).
  scaleY: A number representing the scaling factor along the Y-axis (height).
  scaleZ: A number representing the scaling factor along the Z-axis (depth).

  - A value of 1 means no scaling (original size).

- Values greater than `1` enlarge the element.
- Values less than `1` reduce the size of the element.

Important Points:

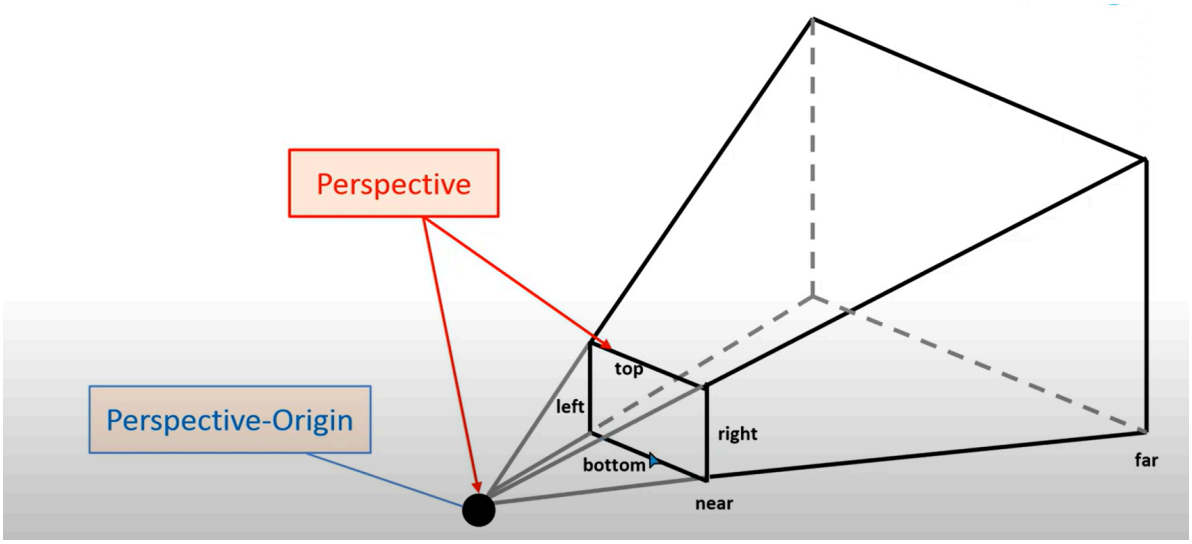
1. **3D Scaling**: Unlike `scaleX()` and `scaleY()`, which only affect width and height, `scale3d()` affects all three dimensions, making it particularly useful for creating depth effects in a 3D space.
2. **Perspective**: For `scale3d()` to have a noticeable visual effect, it's typically used in combination with the `perspective` property or other 3D transformations like rotation.
3. **Visual Effect**: Scaling in the Z-axis can make elements appear closer or further away from the viewer, creating an illusion of depth.

Example:

```
transform: perspective(500px) scale3d(0.5,0.5,2) rotateY(60deg);
```

Perspective Origin:

- The `perspective-origin` property in CSS specifies the position of the viewer in a 3D space. It determines the point from which the user views a 3D transformed element, affecting how the perspective is applied to that element. This property is crucial when creating 3D effects, as it can significantly alter the appearance of transformed elements.



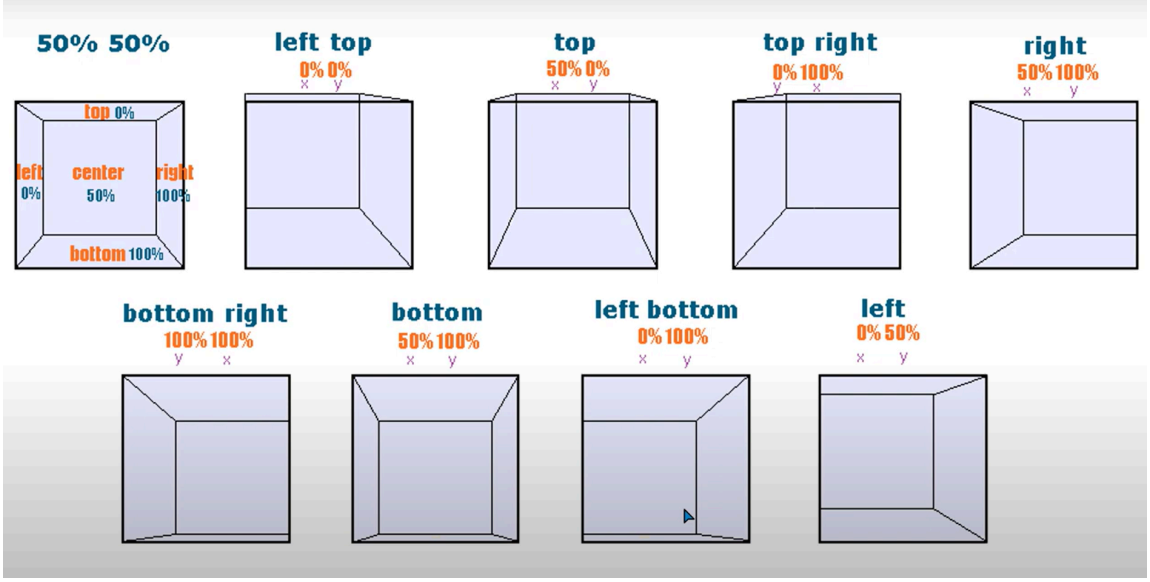
Syntax:

```
perspective-origin: <length> <length>;
```

- The first value defines the horizontal position (X-axis).
- The second value defines the vertical position (Y-axis).
- Both values can be specified using length units like `px` or percentage values (like `50%`).
- You can also use keywords to define common positions:
    - **Horizontal keywords**: `left`, `center`, `right`
    - **Vertical keywords**: `top`, `center`, `bottom`

Key Points:

1. **Default Value**: The default value for perspective-origin is 50% 50%, which means the perspective originates from the center of the element.
2. **Effects on 3D Transformations**: Adjusting the perspective origin can change how an element appears when transformed in 3D. Moving the origin closer to the element can exaggerate the 3D effect, while moving it further away can reduce the effect.
3. **Compatibility**: The perspective-origin property is typically used in conjunction with the perspective property.



Example:

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>TranslateZ Closer and Farther</title>

  <style>
      .container {
          width: 300px;
          height: 300px;
          background-color: gray;
          perspective: 600px;
        perspective-origin: bottom right;
       /* perspective-origin: 100px 150px; */
      /* perspective-origin: 70% 60%; */


      }
```

```
        .box {
            width: 300px;
            height: 300px;
            background-color: yellowgreen;

            transition: transform 2s ease;
        }

        .box:hover  {
          transform: rotateX(60deg);


        }
    </style>

</head>

<body>
    <div class="container">
        <div class="box"></div>
    </div>
</body>




</html>
```

CSS Transform-style:

- The transform-style property in CSS specifies how child elements of a **transformed** element are rendered in 3D space. This property is essential for maintaining the 3D context of child elements when they are transformed, allowing for more complex and visually appealing designs.

Syntax:

```
transform-style: flat | preserve-3d;
```

- flat: This is the default value. It means that the child elements are rendered in 2D space, and their 3D transformations will not be preserved. This can lead to child elements appearing flat and losing their 3D context.
- preserve-3d: When this value is set, the child elements are rendered in 3D space, and their transformations will be preserved. This allows for more complex 3D effects, as the child elements can interact with the 3D transformations of their parent.

Key Points:

1. **Default Behavior:** By default, child elements do not retain their 3D transformations unless preserve-3d is explicitly set.
2. **3D Context:** Using preserve-3d allows you to create intricate 3D designs by keeping the 3D context of child elements, enabling effects like flipping cards, rotating cubes, and more.

Example:

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>TranslateZ Closer and Farther</title>

  <style>
      .container {
          width: 300px;
          height: 300px;
          background-color: gray;
```

```
            perspective: 600px;
            transform-style: preserve-3d;
            transform: rotateX(60deg);

        }

        .box {
            width: 300px;
            height: 300px;
            background-color: yellowgreen;

            transition: transform 2s ease;
        }

        .box:hover  {
          transform: rotatey(60deg);
        /* transform: translateZ(-200px); */


        }
    </style>

</head>

<body>
    <div class="container">
        <div class="box"></div>
    </div>
</body>

</html>
```

Backface-visibility:

- The `backface-visibility` property in CSS controls whether the back side of an element is visible when it is rotated in 3D space. This property is particularly useful when working with 3D transformations, such as flipping cards or creating animations that involve rotating elements.

Syntax:

```
backface-visibility: visible | hidden;
```

- `visible`: The back face of the element is visible when the element is rotated.
- `hidden`: The back face of the element is not visible when the element is rotated. It will be hidden from view.

Key Points:

1. **Default Value:** The default value is `visible`. This means that if you don't specify a value, the back side will be shown when rotated.
2. **Use Case:** Commonly used in 3D transformations, such as flipping cards or creating interactive elements that reveal information on the back when hovered over or clicked.
3. **Compatibility:** To see the effect of `backface-visibility`, you typically need to use it in conjunction with `transform`, especially `rotateY` or `rotateX`.

Example1:

```
<!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{
            width: 200px;
            height: 300px;
            border: 5px solid red;
            margin: 50px auto;
            perspective: 800px;

        }
```

```
        .box{
            width: 100%;
            height: 100%;
            display: flex;
            justify-content: center;
            align-items: center;
            font-size: 3em;
            background-color: aqua;
            transition: transform 2s;
            /* backface-visibility: hidden; */
        }

        .box:hover{
            transform: rotateY(180deg);
        }



    </style>

</head>
<body>

    <div class="container">
        <div class="box">Box</div>
    </div>


</body>
</html>
```

Example2: Flipping the Card

```
<!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{
        width: 200px;
        height: 300px;
        border: 5px solid red;
        margin: 50px auto;
        perspective: 800px;
    }


    .card {
        width: 100%;
        height: 100%;
        border: 2px solid green;

        position: relative;

        /* Preserve 3D for child elements */
        transform-style: preserve-3d;

        /* Smooth transition */
        transition: transform 2s;
    }

    .front, .back{
        width: 100%;
        height: 100%;
        border: 2px solid black;
        position: absolute;

        display: flex;
        justify-content: center;
        align-items: center;

        backface-visibility: hidden;
    }

    .front{
```

```
            background-color: aqua;
        }

        .back{
            background-color: aquamarine;
            transform: rotateY(180deg); /* Position back side */
        }

        .card:hover {
            transform: rotateY(180deg); /* Flip the card on hover */
        }

    </style>


</head>
<body>

    <div class="container">
        <div class="card">
            <div class="front">Front</div>
            <div class="back">Back</div>
        </div>
    </div>


</body>
</html>
```

3D Cube:

```
<!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>

    body{
     width: 100%;
     height: 100vh;
     display: flex;
     justify-content: center;
     align-items: center;
    }

    .container{
     width: 400px;
     height: 400px;
     border: 5px solid red;
     display: flex;
     justify-content: center;
     align-items: center;
     perspective: 1000px;
     perspective-origin: top right;

    }

    .cube{
     width: 200px;
     height: 200px;
     position: relative;
     transform-style: preserve-3d;
     animation: q1 10s infinite ease-in-out;

    }



    .face{
     position: absolute;
     width: 100%;
     height: 100%;
     display: flex;
     justify-content: center;
```

```
 align-items: center;
 font-size: 3em;
 opacity: 0.9; /* Slight transparency */
/* backface-visibility: hidden; */

}

.front{
 background-color: aquamarine;
 transform: translateZ(100px);
}

.back{
 background-color: yellow;
 transform: rotateY(180deg) translateZ(100px);


}

.top{
 background-color: bisque;
 transform: rotateX(90deg) translateZ(100px);


}

.bottom{
 background-color: yellowgreen;
 transform: rotateX(-90deg) translateZ(100px);

}

.left{
 background-color: tomato;
 transform: rotateY(-90deg) translateZ(100px);

}

.right{
 background-color: royalblue;
```

```
        transform: rotateY(90deg) translateZ(100px);

    }


    @keyframes q1 {
        0% {
            transform: rotateX(0) rotateY(0); /* Start position */
        }
        17% {
            transform: rotateY(90deg); /* Show right face */
        }
        33% {
            transform: rotateY(180deg); /* Show back face */
        }
        50% {
            transform: rotateY(270deg); /* Show left face */
        }
        67% {
            transform: rotateX(-90deg); /* Show bottom face */
        }
        83% {
            transform: rotateX(90deg); /* Show top face */
        }
        100% {
            transform: rotateX(0) rotateY(360deg); /* Back to start */
        }
    }
</style>
</head>
<body>


<div class="container">
    <div class="cube">
        <div class="face front">Front</div>
        <div class="face back">Back</div>
        <div class="face left">Left</div>
        <div class="face right">Right</div>
        <div class="face top">Top</div>
```

```
            <div class="face bottom">Bottom</div>
        </div>
    </div>



</body>
</html>
```

Animate.css:

- **Animate.css** is a **CSS library** that provides a collection of ready-to-use animations that you can easily apply to your HTML elements.
- It simplifies the process of adding animations without needing to write complex CSS from scratch.

Features

- **Easy to Use**: You just need to add a class to your HTML elements to apply animations.
- **Responsive**: Animations work well on different screen sizes and devices.
- **Cross-Browser Support**: It supports all modern browsers.
- **Variety of Animations**: Offers a wide range of animations, including fades, bounces, slides, and more.

Getting Started

1. **Include Animate.css in Your Project**:
    - You can either download the CSS file from the Animate.css website or link to it directly from a CDN. Here's how to include it via CDN in your HTML:

2. **Applying Animations**:

    - To use an animation, add the `animate__animated` class along with the desired animation class to the HTML element. For example:

```
<div class="animate__animated animate__bounce">Hello World!</div>
```

Or We can apply these classes using animation property also.

Example:

```
<!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="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min
.css"
  />

  <style>

    h1{

        animation: backInDown 2s infinite;
    }


  </style>


</head>
<body>
<!--
    <h1 class="animate__animated animate__backInDown">Welcome to
Chitkara</h1> -->

    <h1>Welcome to Chitkara</h1>


</body>
```

```
</html>
```

Installing the Animate.css to work offline:

- Install the Nodejs software
- With the installation of NodeJs we also get another software called **npm**(Node package manager).
- Verify the version of both the softwares through the command prompt.
    - **node -n**
    - **npm -v**
- After that install the Animate.css using the following command:
    - `npm install animate.css --save`
- Include the following line inside the head section of the HTML document:

```
 <link rel="stylesheet"
href="./node_modules/animate.css/animate.min.css">
```