Animations and 3D Space

Transitions and Transforms

Transforms:
You can visually manipulate an element using the CSS **transform** property.
It has the following methods:

- translate()
- rotate()
- scale()
- skew()
- matrix()

Translate(): Using this method you can move an element sideways or up and down from its current position. It requires two parameters for the X-axis and the Y-axis.

For example, if you want to move an element 100px from the top and 50px from the left:

```
div{
   transform: translate(50px, 100px);
}
```

If you want to move only in one direction, you can do it using the **translateX()** and **translateY()** methods.

Rotate(): Using this method, you can rotate an element anticlockwise or clockwise from its current position. A positive value will rotate it in a clockwise direction whilst a negative value will rotate it in the anti-clockwise direction. If you want to rotate an element along a specific axis, you can use the **rotateX(), rotateY()** and **rotateZ()** methods which rotate along the X, Y, and Z-axis respectively.

For example if you want an element to rotate an element in clockwise direction with 100 degrees:

```
div {
     transform: rotate(100deg);
}
```

Scale(): You can use this method to increase or decrease the size of an element. It requires two parameters that represent the amount of scaling to be applied in the X-axis and the Y-axis respectively. This method only scales in 2D. The **scale3d()** can be used to scale in 3D.

For example, if you want to scale an element by increasing its width 3 times its original width and increasing its height 5 times its original height:

```
div {
    transform: scale(3,5);
}
```

If you want to increase only the width or only the height of an element, you can use the **scaleX()** and the **scaleY()** methods respectively.

Skew(): If you want to skew an element along the X-axis and the Y-axis, you can use the skew() method. If you want to skew an element only along the X-axis or the Y-axis, use the **skewX()** and **skewY()** methods respectively.

The following example skews an element 20 degrees along the X-axis and 90 degrees along the Y-axis:

```
div {
    transform: skew(20deg, 100deg);
}
```

matrix(): This method combines all the above 2D transform methods and accepts six parameters, which allows you to scale, skew, translate, and rotate.

The parameters are given in this way:

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

```
div {
   transform: matrix(2,0,1,2,10,0);
}
```

Transitions: If you want to change the property of an element over a given period of time, you can use transitions. These are the following transition properties:

- transition
- transition-delay
- transition-duration
- transition-property
- transition-timing-function

To create a transition for an element, you'd need to specify two things:

- The CSS property you want to add a transition to
- The duration of the transition

If you don't specify the duration of the transition, then the transition will have no effect as the default value for the duration is 0.

Look at the example below, where after hovering on the element, its size will scale by a factor of 2 and all this will happen within 2s.

```
div:hover {
   transform: scale(2);
}

div {
   background-color: aquamarine;
   width: 50px;
   transition: 2s ease;
   height: 50px;
}
```


The dimensions of the element begin to increase as soon as we hover over it and it looks like a cool animation. You can create similar effects using the transition property.

Similarly, you can also create a transition effect for multiple properties. You'll have to mention the property names and the duration of each one separately :

```
div {
    transition: width 2s, height 2s;
}
```

You can also adjust the speed curve of the transition with the help of these additional parameters:

- **ease**: the transition will have a slow start, then it'll go fast, and then it'll end slowly
- **linear**: the transition will have a constant linear speed throughout its time duration
- **ease-in**: the transition will have a slow start
- **ease-out**: the transition will have a slow end
- **ease-in-out**: the transition will have a slow start as well as a slow end

Example:

```
 div {
    transition: 2s ease-in;
 }
```

These properties can also be defined separately in the **transition-timing-function** which is used to specify the speed curve of the transition:

```
div {
    transition: 2s;
    transition-timing-function: ease-in;
}
```

The **transition-delay** property will add a delay in the transition, after which the transition will occur.

```
div {
    transition-delay: 2s;
}
```

You can specify the property for which transition is required separately using the **transition-property**:

```
div {
    transition-property: width;
}
```

Similarly, the duration of the transition can be specified separately using the **transition-duration** property:

```
div {
    transition-duration: 4s;
}
```

Creating a simple animation

Simple animations can be created easily using the transition and transform properties. Look at the simple animation below:

Just with basic knowledge of transitions and transforms, animations like this can be created.

```
div:hover {
    transform: translateX(200px) scale(2);
}

div {
    margin: 100px;
    width: 50px;
    background-color: bisque;
    height: 50px;
    transition: 1s ease-in;
}
```

Explanation: When the mouse hovers over the , the **translateX** causes it to move 200px to the right and the **scale** property increases the dimensions of it and all of this occurs with a **transition** of 1s which makes it look like an animation.

If you want the above animation to complete in a specified number of steps, you can use the **steps** property in transition where you need to specify the number of steps and start|end based on your preference.

You can also achieve an animation like this one:

```
div:hover {
   transform: rotate(360deg) translateX(20px) scale(0.75);
   border-radius: 50%;
}

div {
   margin: 100px;
   width: 50px;
   background-color: bisque;
   height: 50px;
   transition: 1s;
}
```

This one also works similarly; as soon as the mouse hovers over the , the **transform:rotate()** property rotates it by 360 degrees along which **translateX** moves it to the right side a little bit and scale reduces its size by a factor of 0.75 and a border-radius of 50% is being applied to it, all within a 1s time frame.

CSS Animation Property

You can also create some advanced animations in CSS using the **animation** property. Each property has a name, which will be used along with the **@keyframe** rule.

@keyframe will define what will happen at specific moments during the animation.

Animation property has some sub-properties which are explained below:

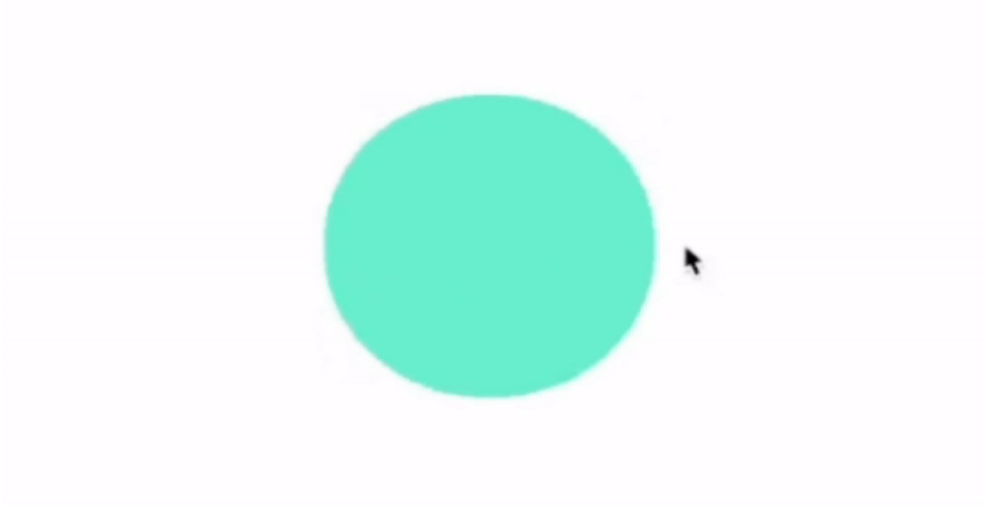
- **animation-name**: it's a name for the keyframe animation
- **animation-duration**: it's the duration of the animation in seconds or milliseconds
- **animation-delay**: it specifies a delay, after which animation will start loading
- **animation-iteration-count**: it's the number of times the animation will be executed
- **animation-timing-function**: it sets a speed curve for the animation;
    - **ease**: the animation will have a slow start, then it'll go fast, then it'll have a slow end.
    - **linear**: the speed of the animation will remain constant throughout its duration
    - **ease-in**: the animation will have a slow start
    - **ease-out**: the animation will have a slow end
    - **ease-in-out**: both the end and the start of the animation will be slow
    - **cubic-bezier(n,n,n,n)**: you can define your own bezier curve using this
- **animation-direction**: it specifies the direction of the animation after each cycle;
    - **normal**: the animation will be played in the normal (default) direction

    - **reverse**: the animation will be played in the reverse direction
    - **alternate**: the animation will be played first in the normal direction, then in the reverse direction
    - **alternate-reverse**: the animation will be played in the reverse direction first, then in the normal direction
- **animation-fill-mode**: it sets how an animation will apply styles to its target, before and after its execution;
    - **none**: it's the default value, meaning that element will not retain any styles before or after the execution of the animation
    - **forwards:** the target element will retain the styles and values set by the last keyframe, which depends upon the values of animation-direction and animation-iteration-count.
    - **backwards:** the target element will retain the styles and values set by the first keyframe, which depends upon animation-direction
    - **both:** In this, the animation will follow both the above rules, so the target element will retain the styles and values in both directions.

@keyframe: It defines what will happen during different stages of animation. This gives you a lot more control over the sequence of the animation as you will define what happens at different sections of the animation. To use @keyframes, you'll need to create one along with a **name**, which is then used by the ***animation-name*** property to match its value with a keyframe declaration.

You'll need to specify the percentage of the time at which the keyframe should occur. The ***from*** and ***to*** keywords represent the initial (start) and the final (end) state of the animation respectively. You can use them instead of writing **0%** and **100%.**
For Example:

```
@keyframes changeShape {
    from {
        border-radius: 0%;
    }
    to {
        border-radius: 50%;
    }
}
div {
    height: 200px;
    width: 200px;
    background-color: aquamarine;
    animation-name: changeShape;
    animation-duration: 1s;
    animation-iteration-count: infinite;
}
```



Explanation: We created a keyframe named *changeShape* and we specified that when the animation starts, the border-radius is 0%, and when the animation ends, it's 50%. When the *animation-name* is matched with the one given to the keyframe, the animation starts to run. As the *animation-duration* is set to 1s, the full animation duration is only 1 second and then it starts to run again. The *animation-iteration-count* is set to **infinite**, which means after completing a cycle of 1s, this animation will continue to run forever.

If we set the *animation-direction* to **alternate**, then the keyframe animation will first run from 0% to 100% and then from 100% to 0%; the transition from rectangle to circle and circle to rectangle will happen in both directions which will look smoother as compared to earlier one.

Instead of **infinite**, if we put a number like 4 in the *animation-iteration-count,* the animation will run only four times in total. You can set the *animation-timing-function* as **ease-in-out,** which will cause the animation to run smoothly during the start and the end.

Instead of only specifying what to do at the start and end of the animation, you can also specify what to do at individual states of the animation. For example, if you want to change the background color of the when the animation reaches 50% of its duration:

```
@keyframes changeShape {
    from {
        border-radius: 0%;
    }
    50% {
        background-color: beige;
    }
    to {
        border-radius: 50%;
    }
}
```

Similarly, you can specify the actions you want to perform at any specific duration of time. You can also write **0%** and **100%** instead of *from* and *to* respectively.

Instead of specifying all the animation properties separately, you can use a shorthand property; *animation* where you can specify all other sub-properties at once. The above animation can also be written like this.

```
animation: changeShape 1s ease-in-out infinite;
```

CSS 3D Transforms

As you know by now, CSS was built to typically style documents, it's not ideal for 3D modelling. But as of 2009, some of the additional 3D features were added in CSS which can be used for some really good 3D animations. We'll explain these by taking examples. The ***perspective*** property is required to activate 3D space. We can apply this like:

```
transform: perspective(400px);
```

OR

```
perspective: 400px;
```

The value of ***perspective*** property determines the ***intensity*** of the 3D effect. You can think of it as a distance from the viewer to the object. The greater the value, the further the distance, the less intense the visual effect. **perspective: 2000px** yields a ***subtle*** 3D effect, as if we are viewing an object from ***far away*** through binoculars whereas, **perspective: 100px** produces a ***tremendous*** 3D effect, like a tiny person viewing a massive object.

This will trigger a 3D space and thus, we can see the element moving and transforming in 3D space. For example:

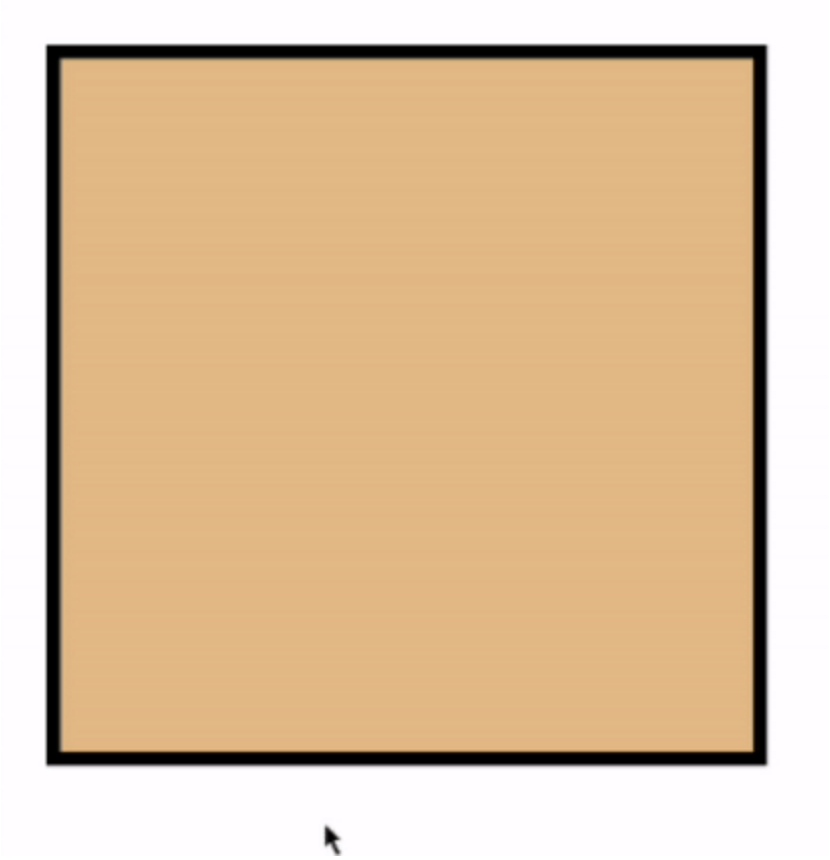
```
<head>
    <style>
        #outer {
            width: 100px;
            height: 100px;
            border: 2px solid white;
            perspective: 400px;
        }

        #inner {
            background-color: burlywood;
            width: inherit;
            height: inherit;
        }

        #inner:hover {
            transform: rotateY(45deg);
        }
    </style>
</head>
<body>
    <div id="outer">
        <div id="inner">

        </div>
    </div>
</body>
```

We have two *** *** tags {inner and outer}, we applied ***perspective:400px*** on the ***outer .*** On hovering over the inner div, it will rotate along the Y-axis and you'll be able to see it:



The 3D transforms use the same `transform` property used for 2D transforms. As you already know about 2D transforms, this will easy for you:

- `rotateX( angle )`
- `rotateY( angle )`
- `rotateZ( angle )`
- `translateZ( tz )`
- `scaleZ( sz )`

Whereas `translateX()` moves an element along the horizontal X axis, `translateZ()` positions it along the Z axis. The rotate functions rotate along the corresponding axis.

In order for children to inherit a parent's perspective, and sustain in the same 3D space, the parent's perspective can passed along with `transform-style: preserve-3d`

We can create a card flipping animation like this:

This is the basic HTML for the card.

```
<body>
    <div class="scene">
        <div class="card">
            <div class="card_face card_face-front">front</div>
            <div class="card_face card_face-back">back</div>
        </div>
    </div>
</body>
```

Everything will happen inside the element with class ***.scene.*** The ***.card element*** will act as the 3D object. Two separate ***.card_face*** elements form the faces of the card.
First we will apply all the necessary ***perspective*** styling to the 3D space, which will contain all the stuff:

```
  .scene {
        width: 200px;
        height: 200px;
        border: 1px solid black;
        margin: 40px 0;
        perspective: 600px;
    }
```

Now, the ***.card*** can be transformed into its parent's (***.scene***) 3d space. We'll use ***position:relative*** so that the card faces are positioned correctly:

```
.card {
        width: 100%;
        height: 100%;
        transition: transform 0.5s;
        transform-style: preserve-3d;
        position: relative;
    }
```

To reset the position of the card faces in 2D space, we used ***position:absolute***:

```
.card_face {
        position: absolute;
        width: 100%;
        height: 100%;
        line-height: 200px;
        color: white;
        text-align: center;
        font-weight: bold;
        font-size: 40px;
}
```

Other styles are there to adjust the position and appearance of the text.

To flip the card on hover, we added a basic 3D transform which will rotate the cards along the Y-axis by 180 degrees:

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

You can refer to the full code here:

```
<style>
    .scene {
        width: 200px;
        height: 200px;
        border: 1px solid black;
        margin: 40px 0;
        perspective: 600px;
    }
        .card {
        width: 100%;
        height: 100%;
        transition: transform 0.5s;
        transform-style: preserve-3d;
        cursor: pointer;
```

```
        position: relative;
    }
        .card:hover {
        transform: rotateY(180deg);
    }
        .card_face {
        position: absolute;
        width: 100%;
        height: 100%;
        line-height: 200px;
        color: white;
        text-align: center;
        font-weight: bold;
        font-size: 40px;
    }
        .card_face-front {
        background: maroon;
    }
        .card_face-back {
        background: royalblue;
        transform: rotateY(180deg);
    }
</style>
 <body>
    <div class="scene">
        <div class="card">
            <div class="card_face card_face-front">front</div>
            <div class="card_face card_face-back">back</div>
        </div>
    </div>
</body>
```

You can create many such animations using 3D transforms. Try creating a 3D Cube, which will test your skill a lot.

EXTRA:
You can refer to this link below to read more about 3D transforms:
https://blog.logrocket.com/the-noobs-guide-to-3d-transforms-with-css-7370aafd9edf/