UNIT-2

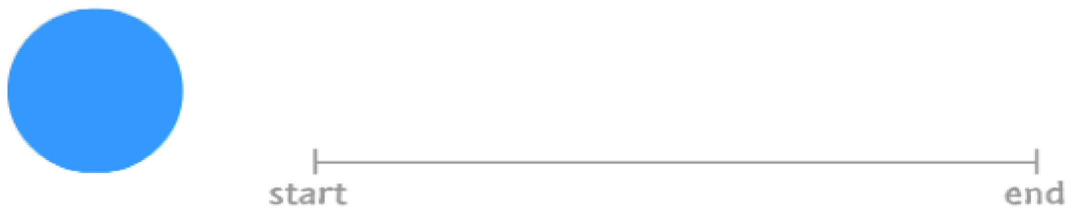
Animation:

What is an Animation?

An Animation lets an element gradually change from one style to another. We can change as many CSS Properties we want, an many times as we want.

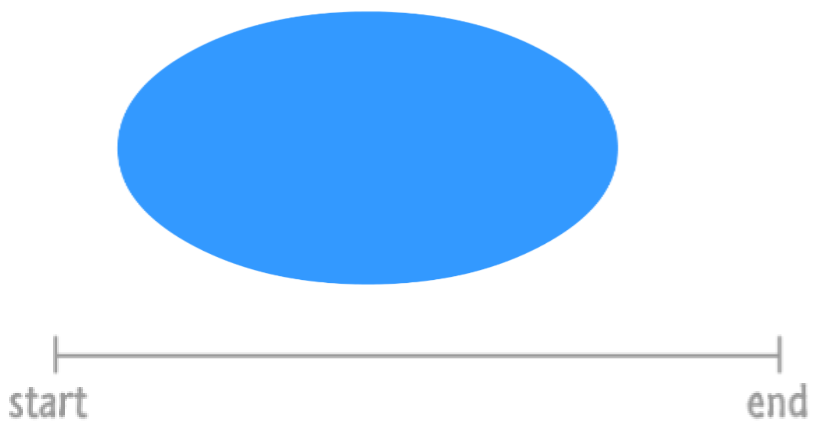
The Start and End States

If visualizing change is an important part of an animation, we need to create some reference points so that we can compare what has changed. Let's call these reference points the **start** state and the **end** state. To better explain what is going on, let's come up with an easy-to-understand example as well.



You start off with a blue circle that is small and located to the left of the screen. At the end state, your blue circle now looks sorta kinda like this:

Animation



Based just on the information you have on what our blue circle looks like in the start and end states, what can you tell is different?

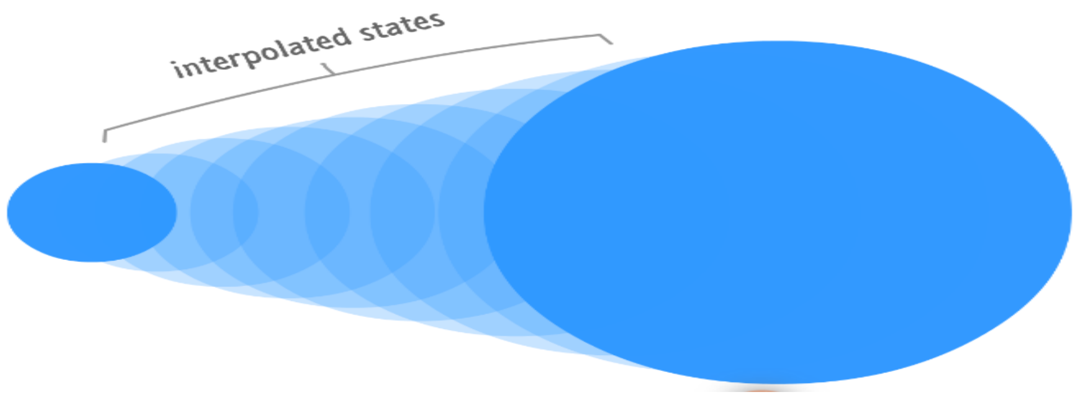
One change is the position. Our blue circle starts off on the left side of the screen. It ends up on the right hand side. Another change is the size. Our circle goes from being small to being much larger.

How do we make an animation out of this? If we were to just play the start and end states repeatedly, what you would see is something that just bounces from left to right very awkwardly. That is pretty turrible. Just turrible. What we need is a way to smooth things out between the start and end states. What we need is a healthy dose of **interpolation**.

Interpolation

Right now, what we have are two discrete states in time. At the beginning, you have your start state. And the end, you have the end state. If you were to play this back, this wouldn't be an animation. In order to make an animation out of what we have, we need a smooth transition that creates all the intermediate states. This creation of the intermediate states is known as **interpolation**.

This interpolation, which occurs over a **period of time that you specify**, would look similar to the following diagram:



You may be wondering who specifies the interpolated states. The answer, which is probably good news, is that your browser or HTML rendering engine will take care of the messy details. All you need to specify is the **starting state**, the **ending state**, and the **duration** over which the transition between the two states needs to occur. Once you have those three things, you have an animation!

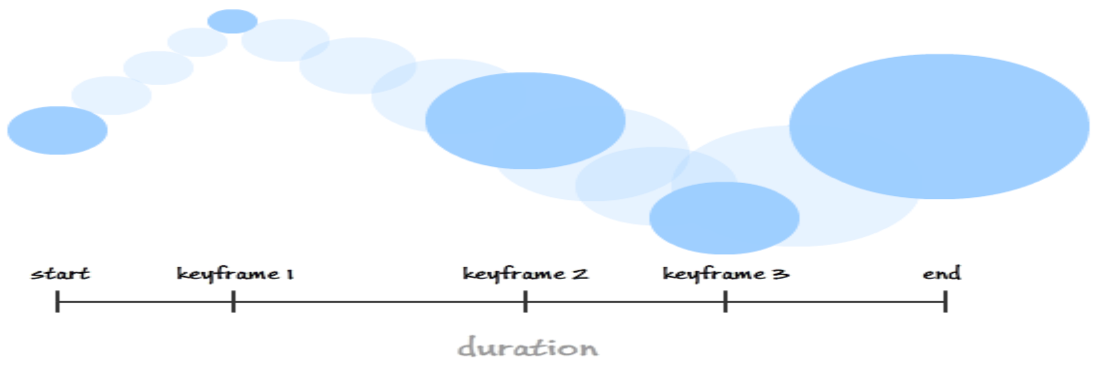
You will later see how adding some other ingredients into the pot such as timing functions (easing functions) can alter how the interpolation works, but we'll get there later. For now, just revel in this simplified generalization of what makes up an animation, put on your best party clothes, and get ready to meet the three flavors of animation that you will end up using.

Animations in HTML

In HTML, there isn't just a single animation implementation (hey, that rhymes!) that you can use. You actually have three flavors of animation to choose from, and each one is specialized for certain kinds of tasks. Let's take a quick look at all three of them and see how they relate to the animation definition you saw in the previous section.

1. CSS Animations (aka Keyframe Animations)

CSS Animations are your traditional animations that on some sort of performance enhancing substance that makes them more awesome. With these kinds of animations, you can define not only the beginning and the end state but also any intermediate states lovingly known as keyframes:

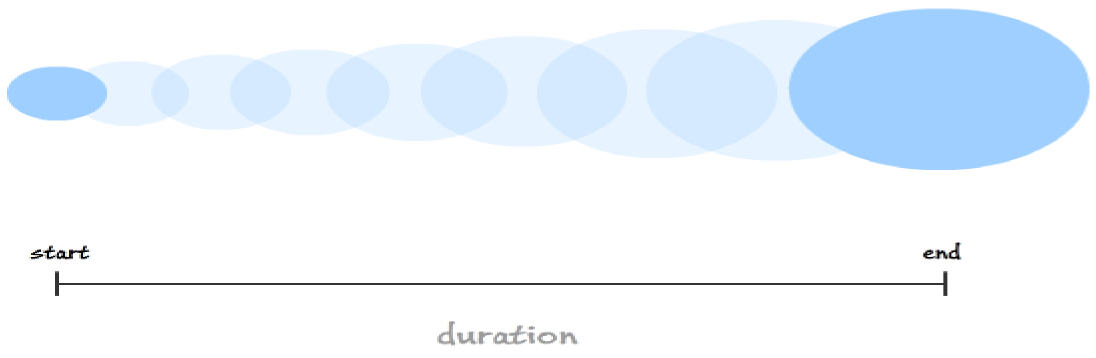


These intermediate states, if you choose to use them, allow you to have greater control over the thing you are animating. In the above example, the blue circle isn't simply sliding to the right and getting larger. The individual keyframes adjust the circle's size and vertical position in ways that you wouldn't see if you simply interpolated between the start and end states.

Remember, even though you are specifying the intermediate states, your browser will still interpolate what it can between each state. Think of a keyframe animation as many little animations daisy chained together.

2. CSS Transitions

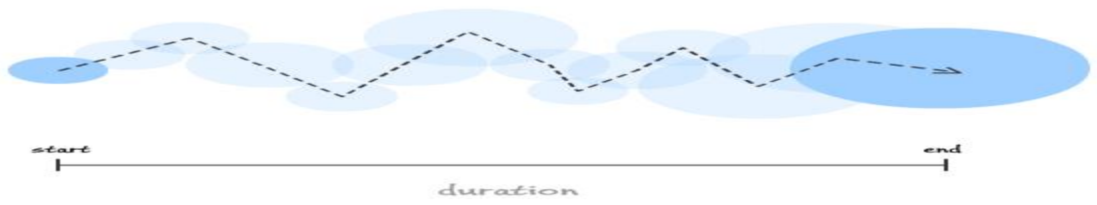
Transitions make up a class of animations where you only define the start state, end state, and duration. The rest such as interpolating between the two states is taken care of automatically for you:



While transitions seem like a watered down, simplified keyframe animation, don't let that trick you. They are extremely powerful and probably my favorite animation technique to use in my projects. You'll see more about them shortly.

3. Scripted / JavaScript Animations

If you want full control over what your animation does right down to how it interpolates between two states, you can use JavaScript:



There are a lot of cool things you can do when you opt-out of the interpolation the browser does for you, and you'll get a good dose of that in the tutorials that look at JavaScript Animations in greater detail.

All about CSS Animations

CSS animations refer to the ability to animate HTML elements using CSS (Cascading Style Sheets). With CSS animations, you can create dynamic and interactive effects on web pages without relying on JavaScript or other scripting languages. CSS animations allow you to apply transitions, keyframe animations, and other effects to elements, bringing them to life and enhancing the user experience.

Here are some key aspects of CSS animations:

1. Transitions: CSS transitions enable smooth and gradual changes in an element's property values over a specified duration. Common properties include color, size, position, opacity, and more. Transitions can be triggered by events such as hovering over an element or clicking on it.

Animation

2. Keyframe Animations: Keyframe animations allow you to define a series of styles at specific points in time, known as keyframes. You can specify how an element should appear and behave at different stages of the animation. Keyframe animations provide greater control over the timing, easing, and sequencing of animation effects.

3. Animation Properties: CSS provides various animation-related properties to control the behavior of animations. These include animation-name (specifying the name of the keyframe animation), animation-duration (setting the duration of the animation), animation-delay (specifying a delay before the animation starts), animation-timing-function (determining the speed and acceleration of the animation), and animation-iteration-count (setting the number of times the animation should repeat).

4.@keyframes Rule: The @keyframes rule is used to define the keyframe animation. It allows you to specify the styles that should be applied at different percentages of the animation's duration. By defining multiple keyframes, you can create complex and customized animations with precise control over the element's appearance and behavior.

5. Transformations: CSS animations often involve transforming an element's shape, position, or size. CSS provides a set of transformation functions such as translate, rotate, scale, and skew, which can be combined to create a wide range of animation effects.

6. Browser Support: CSS animations are supported by modern web browsers, including Chrome, Firefox, Safari, and Edge. However, it's essential to consider browser compatibility and provide fallback options for older browsers that may not support CSS animations.

Creating a Simple Animation

```
<html>

<head>

<title>css</title>

</head>

<style>

.fade-in-out {

  animation-name: fade;

  animation-duration: 2s;

  animation-timing-function: ease-in-out;

  animation-iteration-count: infinite;

}

  @keyframes fade {

  0% {

 Opacity: 0;

  }
```

```
    50% {
      opacity: 1;
    }
    100% {
      opacity: 0;
    }
  }
  </style>
  <body>
  <h1 class="fade-in-out">Hello, World!</h1>
  </body>
  </html>
```

In this example, we have an h1 element with the class "fade-in-out". The animation is applied to this element.

The animation is defined using the @keyframes rule. We named the animation "fade" and set its duration to 2 seconds. The animation-timing-function property is set to "ease-in-out" to create a smooth acceleration and deceleration effect. The animation-iteration-count property is set to "infinite" to make the animation repeat indefinitely.

Within the @keyframes rule, we define three keyframes at different percentages of the animation's duration: 0%, 50%, and 100%. Each keyframe specifies the value of the opacity property at that point in the animation.

At 0% and 100%, the text is completely transparent (opacity 0), creating the fade-out effect. At 50%, the text is fully opaque (opacity 1), creating the fade-in effect.

When you load the HTML file with the above CSS, you'll see the text "Hello, World!" fading in and out continuously.

Feel free to customize this example by modifying the text content, animation duration, timing function, or other properties to create different fade effects and experiment with various animations using CSS.

Detailed Look at the CSS Animation Property

Certainly! Let's take a detailed look at the CSS animation property and its various sub-properties:

The animation property is a shorthand property that allows you to specify multiple animation-related properties in a single declaration. It follows the syntax:

CSS

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

name: Specifies the name of the keyframe animation to be applied. This name corresponds to the @keyframes rule that defines the animation's keyframes.

Animation

duration: Sets the duration of the animation. It specifies how long the animation takes to complete one cycle. You can define it in seconds (s) or milliseconds (ms).

timing-function: Specifies the timing function that controls the pace and acceleration of the animation. Common values include ease (default), linear, ease-in, ease-out, ease-in-out, and various cubic-bezier functions.

delay: Sets a delay before the animation starts. It specifies the time gap between the element being loaded or the animation being triggered and the start of the animation. You can define it in seconds (s) or milliseconds (ms).

iteration-count: Determines the number of times the animation cycles before stopping. You can specify a specific number (1, 2, 3, etc.), infinite for an indefinite number of repetitions, or fractional values (0.5, 1.5, etc.) for partial repetitions.

direction: Defines whether the animation should play forwards (normal), backwards (reverse), alternate between forwards and backwards (alternate), or alternate and play forwards initially (alternate-reverse).

fill-mode: Specifies how the element should be styled before and after the animation. Common values include none (default), forwards, backwards, both, and initial. It determines if the styles defined in the keyframes should apply before the animation starts and after it ends.

It's important to note that you can define multiple keyframes and adjust their properties using the @keyframes rule, which allows for greater control over the animation.

Animation

Here's an example that demonstrates the usage of the animation property:

```
div {

  animation: move 2s ease-in-out 1s infinite alternate;

}

@keyframes move {

  0% {

    transform: translateX(0);

  }

  100% {

    transform: translateX(100px);

  }

}
```

Keyframes

In CSS animations, keyframes define the intermediate stages or snapshots of an animation. They allow you to specify the styles that an element should have at specific points in time during the animation's duration. Keyframes are defined using the @keyframes rule.

- When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.
- To get an animation to work, you must bind the animation to an element.

The following example binds the "example" animation to the <div> element.

The animation will last for 4 seconds, and it will gradually change the background-color of the <div> element from "red" to "yellow":

```
<html>

<style>

div {

  width: 100px;

  height: 100px;

  background-color: red;

  animation-name: example;

  animation-duration: 4s;

}
```

@keyframes example {

```
  from {background-color: red;}
  to {background-color: yellow;}
}
</style>
<body>
<h1>CSS Animation</h1>
<div></div>
<p><b>Note:</b> When an animation is finished, it goes back
to its original style.</p>
</body>
</html>
```

Delay Animation

animation-delay: 2s;

Set How Many Times an Animation Should Run

The animation-iteration-count property specifies the

number of times an animation should run.

The following example will run the animation 3 times before it stops:

animation-iteration-count: 3;

The "infinite“ is used to make the animation continue for ever:

animation-iteration-count: infinite;

<u>Run Animation in Reverse Direction or Alternate Cycles</u>

The animation-direction property specifies whether an animation

should be played forwards, backwards or in alternate cycles.

The animation-direction property can have the following values:

- normal - The animation is played as normal (forwards). This is default
- reverse - The animation is played in reverse direction (backwards)
- alternate - The animation is played forwards first, then backwards
- alternate-reverse - The animation is played backwards first, then forwards

Specify the Speed Curve of the Animation

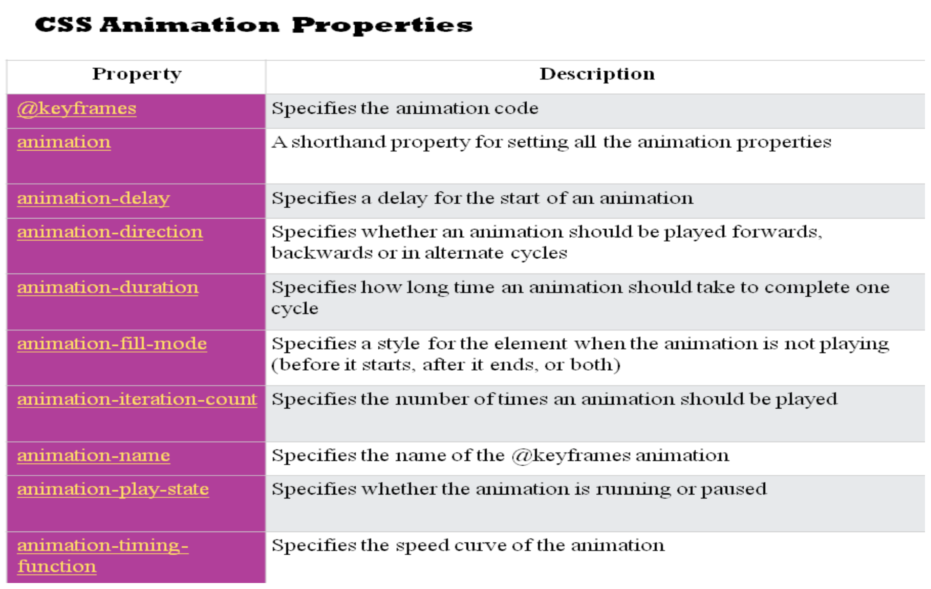
The animation-timing-function property can have the following values:

- ease - Specifies an animation with a slow start,

  then fast, then end slowly (this is default)

- linear - Specifies an animation with the same speed from start to end
- ease-in - Specifies an animation with a slow start
- ease-out - Specifies an animation with a slow end
- ease-in-out - Specifies an animation with a slow start and end
- cubic-bezier (n,n,n,n) - Lets you define your own values in a cubic-bezier function

CSS Animation Properties



CSS Animation Properties

Property	Description
@keyframes	Specifies the animation code
animation	A shorthand property for setting all the animation properties
animation-delay	Specifies a delay for the start of an animation
animation-direction	Specifies whether an animation should be played forwards, backwards or in alternate cycles
animation-duration	Specifies how long time an animation should take to complete one cycle
animation-fill-mode	Specifies a style for the element when the animation is not playing (before it starts, after it ends, or both)
animation-iteration-count	Specifies the number of times an animation should be played
animation-name	Specifies the name of the @keyframes animation
animation-play-state	Specifies whether the animation is running or paused
animation-timing-function	Specifies the speed curve of the animation

Declaring Multiple Animations

To declare multiple animations in CSS, you can use the @keyframes rule to define each animation and then assign those animations to different elements using the animation property. Here's an example:

Animation

CSS

```
@keyframes animation1 {

  /* Animation keyframes */

}

@keyframes animation2 {

  /* Animation keyframes */

}

.element1 {

  animation: animation1 1s infinite;

}

.element2 {

  animation: animation2 2s alternate;

}
```

In this example, two animations named animation1 and animation2 are declared using the @keyframes rule. Each animation is defined by specifying different keyframes at different percentages of the animation duration. Then, the .element1 class is assigned the animation1 animation with duration of 1 second and an infinite repetition. The .element2 class is assigned the animation2 animation with duration of 2 seconds and an alternating direction.

Animation

You can add as many @keyframes blocks as you need for your animations and assign them to different elements using their respective class selectors or IDs.

All about CSS Transitions

CSS transitions allow you to animate property changes smoothly over a specified duration. They provide a simple way to add animation effects to elements without the need for complex JavaScript or animation libraries. Here's a comprehensive guide to CSS transitions:

1.Transition Properties:

CSS transitions work by animating specific properties of an element. Some common properties that can be transitioned include:

1. background-color: Change the background color of an element.

2. Color: Modify the text color.

3. Width, height: Adjust the dimensions of an element.

4. Opacity: Control the transparency of an element.

5. Transform: Apply 2D or 3D transformations, such as scaling, rotating, and translating.

2. Syntax: The syntax for declaring a CSS transition is as follows:

```
selector {

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

}

1. selector: The CSS selector targeting the element(s) to be transitioned.

2. property: The CSS property to be animated. You can specify multiple properties separated by commas.

3. duration: The duration of the transition. It can be specified in seconds (s) or milliseconds (ms).

4. timing-function (optional): The timing function defines the acceleration curve of the transition. Common options are linear, ease, ease-in, ease-out, ease-in-out, and cubic-bezier().

5. Delay (optional): The delay before the transition starts.

3. Transform Transitions:

The transform property allows you to apply various transformations like scaling, rotating, and translating. Transitions can be applied to transform properties as well. For example:

```
box {

  transition: transform 0.5s;

}



.box:hover {

  transform: rotate(45deg) scale(1.2);
}
```

4. Transition Shorthand:

Instead of specifying each individual property, you can use the transition shorthand property to declare multiple transitions at once. For example:

```
.element {
  transition: property1 duration1, property2 duration2;
}
```

Adding a Transition, Looking at Transitions in Detail

1. Transition Property:

The transition property is used to specify which CSS properties should be animated and how. It is a shorthand property that combines multiple individual transition properties. The syntax is as follows:

transition: property duration timing-function delay;

1. Property: Specifies the CSS properties to be transitioned. Multiple properties can be separated by commas.

2. duration: Defines the duration of the transition in seconds (s) or milliseconds (ms).

3. timing-function (optional): Specifies the acceleration curve of the transition. It controls the speed of the animation over time.

Common options include linear, ease, ease-in, ease-out, ease-in-out, and cubic-bezier().

4. Delay (optional): Specifies the delay before the transition starts.

2. Transition Timing Functions:

Timing functions define the speed curve of the transition. They control how the animation progresses over time. Here are some commonly used timing functions:

1. linear: The transition occurs at a constant speed.

2. ease: The default timing function that starts slow, accelerates in the middle, and slows down again.

3. ease-in: The transition starts slowly and accelerates as it progresses.

4. ease-out: The transition starts fast and slows down as it completes.

5. ease-in-out: A combination of ease-in and ease-out, where the animation starts slowly, accelerates, and then slows down towards the end.

6. cubic-bezier(): Allows you to define custom timing functions using Bézier curves.

3. Multiple Property Transitions:

You can apply transitions to multiple properties by separating the with commas in the transition property. Here's an example:

Animation

```
.box {
  transition: background-color 0.5s ease-in-out, width 0.3s linear;
}
.box:hover {
  background-color: red;
  width: 200px;
}
```

4. Transition Delays:

You can add a delay before the transition starts by specifying the delay value in the transition property.

The Longhand Properties

Longhand properties, in the context of web development and CSS (Cascading Style Sheets), refer to individual CSS properties that are used to define specific aspects of an element's style. These properties provide granular control over various visual and layout characteristics of HTML elements. Longhand properties are often contrasted with shorthand properties, which allow you to set multiple related properties using a single declaration.

Here are some commonly used longhand properties in CSS:

font-family: Specifies the font family or font stack for text.

font-size: Sets the size of the font.

font-weight: Defines the weight or thickness of the font.

Animation

font-style: Specifies the style (normal, italic, or oblique) of the font.

color: Sets the color of text.

background-color: Defines the background color of an element.

padding: Specifies the padding (space) within an element.

margin: Sets the margin (space) around an element.

border: Defines the properties of an element's border, such as width, style, and color.

width and height: Specify the dimensions of an element.

display: Determines how an element is displayed (e.g., block, inline, inline-block).

position: Specifies the positioning scheme of an element (e.g., static, relative, absolute, fixed).

top, right, bottom, left: Used in conjunction with position to precisely position an element.

float: Positions an element to the left or right of its container, allowing text and other elements to wrap around it.

text-align: Sets the horizontal alignment of text within an element.

text-decoration: Controls the decoration of text (e.g., underline, overline, line-through).

text-transform: Modifies the capitalization of text (e.g., uppercase, lowercase, capitalize).

text-overflow: Determines how overflowing text is handled within an element.

line-height: Specifies the height of a line of text.

opacity: Defines the transparency level of an element.

Animation

These are just a few examples, and there are many more longhand properties available in CSS. They offer precise control over the visual appearance and layout of HTML elements, allowing web developers to create customized and visually appealing web pages.

Longhand Properties vs. Shorthand Properties

Longhand properties and shorthand properties are two ways of defining CSS properties in web development.

1. Longhand properties: Longhand properties are individual CSS properties that allow you to specify specific aspects of an element's style. They provide granular control and allow you to define properties like font-size, color, padding, margin, etc., individually.

Example:

font-family: Arial, sans-serif;

font-size: 16px;

color: #333333;

padding-top: 10px;

padding-right: 20px;

padding-bottom: 10px;

padding-left: 20px;

As you can see, each property is declared separately, which gives you precise control over each aspect of the element's style. However, using longhand properties can be verbose and time-consuming, especially when multiple properties need to be set.

2. Shorthand properties: Shorthand properties allow you to set multiple related properties using a single declaration. They provide a more concise way of defining styles. Shorthand properties typically combine several longhand properties into a single declaration

Example:

font: 16px Arial, sans-serif;

color: #333333;

padding: 10px 20px;

In this example, the font property combines font-size and font-family into a single declaration. Similarly, the padding property combines padding-top, padding-right, padding-bottom, and padding-left into a single declaration.

Shorthand properties can help reduce code duplication, improve readability, and make styling more efficient. However, they may not provide the same level of control as longhand properties, especially when you need to override specific values within the shorthand declaration.

Working with Multiple Transitions

Transition keywords:

- **transition:** This keyword can be used with a CSS property in inline, internal, or external CSS. This needs the **property (transition-property)** which will be transitioned, the **time duration (transition-duration)** of the transition, **timing(transition-timing-function)** function of the transition. We can give those values individually to the property or we can use the shorthand technique to add all of them at the same time.
- **transition-property:** This is used to specify the properties to be transitioned.
- **transition-duration:** This is used to specify the time duration for which the properties will be transitioned.
- **transition-timing-function:** This is used to specify the time duration for which the properties will be transitioned.

Example 1: In the below-given code, we have added transitions to transform the color, border, padding-top and padding-bottom, and font size using the **shorthand transition** form.

<!DOCTYPE html>

Animation

```
<html lang="en">

<head>

        <style>

                h3 {

                        color: brown !important;

                }

              .container {

                        display: flex !important;

                }

                        div {

                        font-family: "Lucida Sans", "Lucida Sans Regular";

                        font-size: 1rem;

                        margin: 2rem;

                        justify-content: center;

                        display: flex;

                        border: 10px solid green;

                        width: 18rem;

                        height: 7rem;

                        padding-bottom: 20px;

                        padding-top: 20px;

                        transition: color 1s ease-out, padding-top 1s ease-out,

                                padding-bottom 1s ease-out, font-size 2s ease-out;

                }
```

Animation

```
                    div:hover {
                     color: rebeccapurple;
                     border: 10px solid brown;
                     padding-top: 100px;
                     padding-bottom: 30px;
                     font-size: 1.8rem;
                }
        </style>
</head>
<body>
        <h1 style="color: green; margin: 2rem;">
                PESIAMS
        </h1>
        <h3 style="margin: 2.2rem; margin-top: -2rem">
                How to have multiple CSS
                transitions on an element?
        </h3>
        <div>PESIAMS</div>
</body>
</html>
```

Animation

Animation