UNIT - 2

CSS AND ANIMATIONS

CSS : Cascading Style Sheets

What is CSS?

CSS works with HTML, but it's not HTML. It's a different language altogether. While HTML provides structure to a document by organizing information into headers,paragraphs, bulleted lists, and so on, CSS works hand-in-hand with the web browser to make HTML look good.

- CSS is the language we use to style a Web page.
- CSS can control the appearance of web pages, including font size, color, margins, padding, and much more.
- CSS describes how HTML elements are to be displayed on screen.
- CSS saves a lot of work. It can control the layout of multiple web pages all at once.

CSS allows web designers and developers to separate the presentation of content from its structure and behavior, which makes it easier to maintain and update a website.

The style definitions are normally saved in external **.css** files. With an external stylesheet file, you can change the look of an entire website by changing just one file.

Levels of Style Sheets:

CSS has several levels, each with increasing levels of functionality and complexity:

CSS1 was the first official version of CSS, released in 1996. It introduced the basic concepts of CSS, such as selectors, properties, and values. CSS1 also introduced the box model and layout features, such as floats and positioning.

CSS2 was released in 1998 and added many new features, including support for media types, such as print and screen, as well as support for tables and generated content. CSS2 also introduced the concept of absolute, relative, and fixed positioning, as well as z-index and overflow.

CSS3 is the latest version of CSS and it is divided into several modules, each adding new features and functionality. Some of the key features of CSS3 include :

- Many new selectors, including attribute selectors ,
- New properties to control box sizing, box shadows, and rounded corners.
- New layout features, such as flexbox and grid
- Support for transitions and animations , media queries

Overall, CSS3 is a significant improvement over the previous versions of CSS, and it has enabled developers to create more dynamic and visually appealing web pages than ever before.

Defining a Style :

Eg : p { color: red; font-size: 1.5em; }

This style simply says, "Make the text in all paragraphs—marked with <p> tags—red and 1.5 ems tall." (An em is a unit of measurement that's based on a browser's normal text size) Even a simple style like this example contains some key elements:

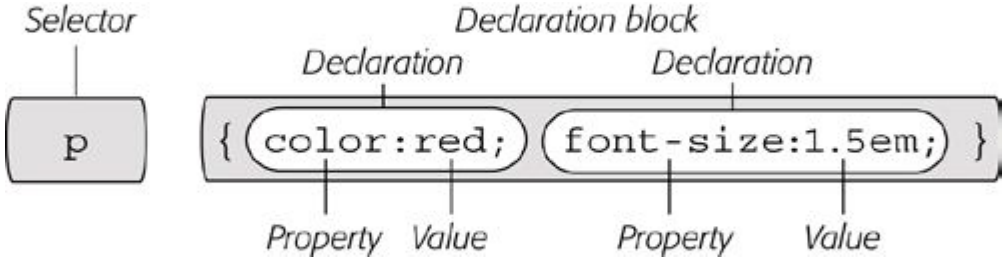
Selectors: Selectors are used to target HTML elements in order to apply styles to them. There are many types of selectors in CSS, including class selectors, ID selectors, attribute selectors, and pseudo-classes.

Declaration Block: The code following the selector includes all the formatting options you want to apply to the selector. The block begins with an opening brace ({) and ends with a closing brace (}).

Declaration: Between the opening and closing braces of a declaration block, you add one or more declarations, or formatting instructions. Every declaration has two parts, a property and a value. A colon separates the property name and its value, and the whole declaration ends with a semicolon

Property: CSS offers a wide range of formatting options, called properties,indicating a certain style effect.

Value:assigning a value to a CSS property



<u>Style Specification Formats:</u> There are three ways to specify styles in CSS:

1. Inline Style:

Inline styles are applied directly to an HTML element using the style attribute. It is useful when you want to apply a style to only one specific element. Here is an example:

```
<p style="color: red; font-size: 20px;">This text is red and 20 pixels in size.</p>
```

2. Internal Style Sheet :

An internal style sheet is a collection of styles that's part of the web page's code. It always appears between opening and closing HTML <style> tags in the page's <head> portion. Here's an example:

```
<!DOCTYPE html>
<html>
<head>
<style>
      h1 {
        color: blue;
        font-size: 36px;
      }
</style>
</head>
<body><h1>Hello World</h1></body>
</html>
```

3. External Style Sheets :

An external style sheet is nothing more than a text file containing all your CSS rules. It never contains any HTML code. So it ***doesn't include the <style> tag***. In addition, file will be saved with the extension .**css.**

We can attach a style sheet to a web page two methods :

- Linking a Style Sheet by Using HTML **<link> tag**
- Linking a Style Sheet by Using CSS **@import directive**

Linking a Style Sheet by Using HTML : The most common method of adding an external style sheet to a web page is to use the HTML **<link> tag.**

Eg : Using <link> tag : **<link rel="stylesheet" href="css/styles.css">**

- rel="stylesheet" indicates the type of link—in this case, a link to a style sheet.
- href points to the location of the external CSS file

Linking a Style Sheet by Using CSS : CSS includes a built-in way to add external style sheet using **@import directive.** You add the directive inside of an HTML <style> tag .

Eg :
```
<style>
        @import  url(css/styles.css);
        @import  url(css/form.css);


</style>
```

To make the connection to the external CSS file, you use **url** instead of href and enclose the path in parentheses.

we can include multiple external style sheets by using more than one @import:

Selectors: Identifying What to Style

Selectors are used to target HTML elements and apply styles to them. There are many types of selectors in CSS, including:

a) Tag Selectors : Page-Wide Styling

Tag selectors are a type of CSS selector that targets HTML elements based on their tag name. This type of selector can be useful for applying page-wide styles to a specific type of element.

Here's an example: To apply some styles to all the paragraphs (<p>) on your website.

```
p { font-family: Arial, sans-serif; font-size: 16px; color: #333; }
```

b) Class Selectors: Pinpoint Control

Class selectors are a type of CSS selector that targets HTML elements based on their class attribute. This type of selector can be used to apply styles to specific elements on a page that share a common class, while leaving other elements unaffected.

Few rules to keep in mind when naming a class:

- All class selector names must begin with a period ' . ' .
- CSS permits only letters, numbers, hyphens, and underscores in class names.
- After the period, the name must always start with a letter. i.e., .9lives isn't a valid class name, but .crazy8 is.
- Class names are case sensitive. For example, CSS considers .SIDEBAR and .sidebar two different classes.

Here's an example of Class Selector:

```
<style>
        .primary {
                background-color: #f44336;
                color: #fff;
        }
</style>
<button>Click me</button>
<button class="primary">Sign up</button>
<button class="primary">Learn more</button>
<button>Get started</button>
```

c) ID Selectors: Specific Page Elements

CSS reserves the ID selector for identifying a unique part of a page, like a banner, navigation bar, or the main content area. Just like a class selector, you create an ID by giving it a name. To apply style properties to particular id's a pound or hash symbol will be used before the id name.

Here's an example:

```
<style>
#head1 {
        background-color: #f44336;
        color: #fff;
}
</style>
<h1 id="head1">My Website</h1>
<h1> welcome to the portal</h1>
```

d) Styling Groups of Tags :

Group selectors let you apply a style to multiple selectors at the same time. Where each selector will be written after adding a comma. So the group of selectors will have the same properties which is defined.

Here's an example:

```
h1, h2, h3, h4, h5, h6 { color: #F1CD33; }          // grouping all tags
h1, p,.copyright, #banner { color: #F1CD33; }  // grouping tags,class and id
```

e) The Universal Selector (Asterisk) :

An asterisk (*) is universal selector shorthand for selecting every single tag. The asterisk, is a much shorter way to tell CSS to select all HTML tags on the page:

Here's an example:

```
 * { font-weight: bold; }                  // selects all html tags
.banner * { font-weight: bold; }        // selects all tags inside the tag with class='banner'
```

f) Styling Tags Within Tags :

CSS allows to format the html page using the Descendent selectors, Descendent selectors will provide advantage of the HTML family tree by formatting tags differently when they appear inside certain other tags or styles.

To create a descendant selector, you simply list the parent element followed by a space, and then list the child element. Here's an example:

```
<h1>Header
    <p>A paragraph of <strong>important</strong>text.</p>
</h1>
h1 strong { color: red; }   // the text 'important' will be in red
```

g) Pseudo-Classes and Pseudo-Elements :

Pseudo-classes and pseudo-elements are special selectors in CSS that allow you to target specific elements based on their state or position within the document.

The single colon ':' is used for both pseudo-classes and pseudo-elements in CSS1 and 2. However, in CSS3, the double colon '::' notation was introduced specifically for pseudo-elements. However, for backward compatibility, some pseudo-elements can still be written with a single colon : in CSS level 1 and 2.

Pseudo-classes target elements based on their state or user interaction.

Examples of pseudo-classes include ':hover', ':active', ':focus', and ':nth-child()' ..etc

Below are the four different states based on how a visitor has interacted with that link **<a>.**

1) **a:link** selects any link that your guest hasn't visited yet, while the mouse isn't hovering over or clicking it.
   This style is your regular, unused web link.

2) **a:hover** lets you change the look of a link as your visitor passes the mouse over it.

3) **a:active** lets you determine how a link looks as your visitor clicks.

4) **a:visited** is a link that your visitor has clicked before, according to the web browser's history.

<u>More Pseudo-Classes and Pseudo-Elements :</u>

':focus' : targets an element when it has focus.

```
input:focus {  border: 2px solid blue;  }
```

'::before' : adds content before the selected element.

```
h1::before { content: " This is added at the beginning "; }
```

'::after' : adds content after the selected element.

```
h1::after { content: " This is added at the beginning "; }
```

'::first-letter' : targets the first letter of the selected element.

```
p::first-letter { color: red; }
```

'::first-line' : targets the first line of the selected element.

```
p::first-line { color: blue; }
```

'::selection' : refers to items that a visitor has selected on a page.

```
::selection { background-color: red; }
```

h) Attribute Selectors :

Attribute selectors are a type of CSS selector that allow you to target HTML elements based on the presence or value of an attribute. They are denoted using square brackets [] containing the name of the attribute and optionally, a value or operator. Here are some examples of attribute selectors:

[attribute] : targets elements that have the specified attribute, regardless of its value.

```
img[alt] { border: 4px solid black; }
```

[attribute=value] : targets elements that have the specified attribute with the exact value.

```
input[type="submit"] {  background-color: green; }
```

[attribute*=value] : targets elements that have the specified attribute with a value **containing the specified substring.** Eg: a[href*="co.in"] { color: red; }

[attribute^=value] : targets tags that have the specified attribute with a value **starting with the specified string.** Eg: a[href^="http://"] { font-weight: bold; }

[attribute$=value] : targets elements that have the specified attribute with a value **ending with the specified string.** Eg: a[href$=".pdf"] { background-color: yellow; }

i) Child selectors (>) :

Child selectors are a type of CSS selector that allow you to target elements that are direct children of a specific parent element. They are denoted using the **'>'** symbol, which is placed between the parent element and the child element. Here is an example of a child selector:

ul > li { color: blue; }

<u>CSS3 also includes some very specific pseudo-classes for selecting child elements :</u>

':first-child' : targets the first child element of its parent. (alternate '**:first-of-type**')

ul li:first-child { font-weight: bold; }

':last-child' : targets the last child element of its parent. (alternate **':last-of-type'**)

ul li:last-child { color: red; }

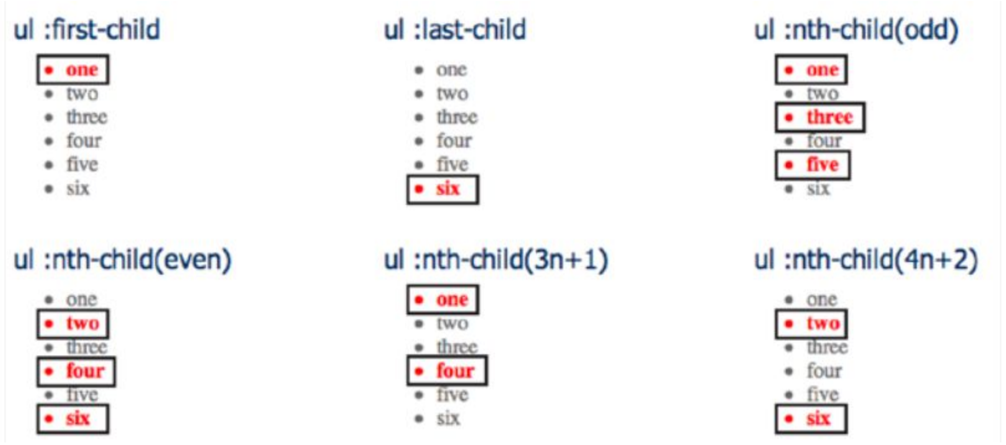
':nth-child(n)' : targets the n-th child element of its parent, where n can be a number, a keyword, or a formula. (alternate **':nth-of-type'**)

ul li:nth-child(odd) { background-color: lightgray; }

ul li:nth-child(even) { background-color: red; }

ul li:nth-child(3n+2) { background-color: green; }

Here, 3n means every third element, while +2 indicates which element to start at



j) Adjacent Sibling Selector (+) :

The adjacent sibling selector is used to select an element that is directly after another specific element. Sibling elements must have the same parent element, and "adjacent" means "**immediately following**".

```
Eg:             h2 + p {  color: blue; }
                <div><h2>Hai</h2>
                        <p>Paragraph 1 </p>    // this will be in blue
                        <p>Paragraph 2 </p></div>
```

k) General Sibling Selector (~) :

The general sibling selector selects all elements that are next siblings of a specified element.

For Eg: while h2 + p selects a single <p> tag that immediately follows a <h2> tag,

h2 ~ p selects all <p> tags that are siblings of <h2> tag

```
                        h2 ~ p {  color: blue; }
                <div><h2>Hai</h2>
                        <p>Paragraph 1 </p>              // text in blue
                        <p>Paragraph 2 </p> </div>    // text in blue
                        <p>Paragraph 3 </p>
```

The ':not()' Selector : targets the elements that do not match a particular selector. It takes a single argument, which is the selector for the elements that you want to *exclude*.

Here's an example:

```
img:not(.logo) {   border: 1px solid black; }
```

This selector selects all img tags on a page, except for those with the class '.logo'

<u>Style Inheritance :</u>

HTML tags can inherit CSS properties from their ancestors.inheritance is the process by which some CSS properties applied to one tag are passed on to nested tags.

For Example :

```
<html><head>
        <style>   body { color:blue; } </style> </head>
<body> <h1> Sample Text1 </h1>
          <p> Inheritance Sample </p>
</body></html>
```

Here **h1 and p** tags will inherit the color property from its ancestor **body** tags property.

Property Value Forms :

CSS Property Value Forms refer to the different ways values can be assigned to CSS properties. CSS offers various forms for specifying property values, allowing for flexibility and customization in styling. Here are some common CSS property value forms:

i) Length Values: Length values are used to specify measurements, such as widths, heights, margins, and paddings. Examples include ***px (pixels), em (relative to the font size), rem*** (relative to the root font size), vh (viewport height), and vw (viewport width). Eg: width: 200px; The *width property* sets the width of an element to length *value 200px*

ii) Keyword Values: Many CSS properties accept predefined keywords as their values. For example, the display property can take values like ***block***, ***inline***, ***or none.*** These keywords represent specific display behaviors for elements.
Eg: display: block; The *display property* takes the *keyword value block*, specifying that an element should be rendered as a block-level element.

iii) Percentage Values: Percentage values are relative to another value, typically the parent element's size. They are commonly used for width, height, or positioning properties. For instance, setting a width of ***50%*** means the element will take up ***50%*** of its parent's width. Eg: height: 50%; The *height property* sets the height of an element to *50%*

iv) Color Values: CSS supports various color value formats, including color names (e.g., red, blue), hexadecimal values (e.g., #ff0000 for red), RGB values (e.g., rgb(255, 0, 0)), and HSL values (e.g., hsl(0, 100%, 50%) for red).
Eg: color: red; The *color property* sets the text color to *red* using a color value.

v) URL Values: Certain properties, like background-image or font-face, accept URL values to specify the location of external resources, such as images or fonts. Eg : background-image: url('image.jpg');
The *background-image property* uses the *URL value 'image.jpg'* to specify the background image for an element.

vi) Multiple Values: Some properties allow multiple values to be specified, separated by spaces or commas. For example, the font property can take values for font-size, font-family, and font-weight in a single declaration.
Eg: font: 16px Arial, sans-serif; The *font property* sets the font size, font family, and fallback font family for an element, using *multiple values* in a single declaration.

vii) Function Values: CSS provides built-in functions that allow for advanced value manipulation. For example, the calc() function enables mathematical calculations, and the linear-gradient() function generates gradient backgrounds. Eg: background-image: linear-gradient(to right, red, blue);
The *background-image property* uses the *linear-gradient() function* to create a linear gradient background from red to blue from left to right.

Font properties :

CSS font properties are used to control the appearance of text on a web page. Here are some commonly used CSS font properties along with examples:

1. font-family : Specifies the font family or typeface to be used. Multiple fonts can be listed in order of preference.
For example: p { font-family: Arial, sans-serif; }
This sets the font of all <p> elements to Arial, and if Arial is not available, it falls back to a generic sans-serif font.

2. font-size : Sets the size of the font. It can be specified in pixels (px), em units, or percentages.
For example: h1 { font-size: 24px; }
This sets the font size of all <h1> headings to 24 pixels.

3. font-weight : Specifies the thickness or boldness of the font. It can be set to normal, bold, or numeric values ranging from 100 to 900.
For example: span { font-weight: bold; }
This makes all <span> elements appear in bold.

4. font-style : Controls the style of the font, such as normal or italic.

For example: p { font-style: italic; }

This makes all <p> elements appear in italic.

5. font-color : Sets the color of the text. It can be specified using color names, hexadecimal codes, RGB values, or HSL values. For example: p { color: #ff0000; }

This applies a red font color (#ff0000) to all <p> elements

6. text-decoration : Determines if any decorations, such as underline or line-through, should be applied to the text.

For example: a { text-decoration: none; }

This removes the underline from all links (<a> elements).

7.line-height: Sets the height of a line of text, controlling the spacing between lines. It can be specified as a number, unit value, or a percentage.

For example: p { line-height: 1.5; }

This sets the line height of all <p> elements to 1.5 times the font size.

8. letter-spacing: Adjusts the spacing between individual characters. It can be set in pixels (px) or other unit values.

For example: h2 { letter-spacing: 2px; }

This adds 2 pixels of space between each character in all <h2> headings.

9. text-transform: Changes the capitalization of the text. It can be set to uppercase, lowercase, capitalize (first letter of each word capitalized), or none. For example: .title { text-transform: uppercase; }

This transforms all text within elements with the class "title" to uppercase.

10. text-align: Specifies the alignment of the text within its container. It can be set to left, right, center, or justify.

For example: .centered { text-align: center; }

This centers the text within elements that have the class "centered".

List properties:

CSS list properties are used to style and customize the appearance of lists on a web page. Here are some commonly used CSS list properties along with examples:

1. list-style-type: Specifies the type of marker or bullet used for list items. It can be set to various values such as none, disc, circle, square, decimal, lowercase letters, uppercase letters, and more.

For example: ul { list-style-type: square; }

This sets the marker for unordered lists (<ul>) to square shapes.

2. list-style-position: Determines the position of the marker or bullet relative to the list item text. It can be set to inside or outside. For example: ol { list-style-position: inside; }

This positions the numbering marker inside the list items of ordered lists (<ol>).

3. list-style-image: Allows you to use a custom image as the marker or bullet for list items. It specifies the URL of the image file. For example: ul { list-style-image: url('bullet.png'); }

This sets a custom bullet image for unordered lists, using the image file "bullet.png".

5. list-style : A shorthand property that combines the *three list properties* (list-style-type, list-style-position, and list-style-image) into a single declaration.

For example: ul { list-style: square inside url('bullet.png'); }

This sets the marker for unordered lists to square shapes, positions it inside the list items, and uses a custom bullet image.

Color Property :

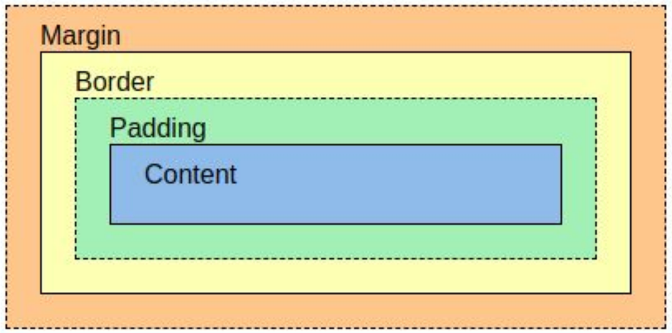
Color is a property that allows you to control the color of text, backgrounds, and other elements on a web page. It offers various ways to specify colors, including predefined color names, hexadecimal codes, RGB values, and HSL values.

- **Predefined Color Names:** CSS provides a set of predefined color names such as red, blue, green, and so on
- **Hexadecimal Codes:** Hexadecimal codes represent colors using a combination of six characters, consisting of numbers (0-9) and letters (A-F). eg: #ff0000 //red color
- **RGB Values:** RGB stands for Red, Green, and Blue. It allows you to define a color by specifying the intensity of each primary color component. Values range from 0 to 255. eg: rgb(255,0,0) // red color
- **HSL Values:** HSL stands for Hue, Saturation, and Lightness. It provides a way to define colors based on their hue, saturation, and lightness values. Hue is represented as an angle from 0 to 360, while saturation and lightness are percentages ranging from 0% to 100%. eg: hsl(0,100%,50%) //red color

<u>**CSS-Box Model**</u> :

To a browser, any tag is a box with something inside it—text, an image, or even other tags containing other things, Surrounding the content are different properties that make up the box:

- **padding** is the space between the content and the content's border. Padding is what separates a photo from the border that frames the photo.
- **border** is the line that's drawn around each edge of the box. You can have a border around all four sides, on just a single side, or any combination of sides.
- **background-color** fills the space inside the border, including the padding area
- **margin** is what separates one tag from another. The space that commonly appears between the tops and bottoms of paragraphs of text on a web page



Margins and Padding :

Both margins and padding add space around content. You use these properties to separate one element from another.Padding and margin function similarly, and unless you apply a border or background color, you can't really tell whether the space between two tags is caused by padding or by a margin.

But if you have a border around an element or a background behind it, then the visual difference between the two properties is significant. Padding adds space between the content and the border, and keeps the content from appearing cramped inside the box; it also includes the background area, so the space taken up by padding might be empty of content (like text or a photo), but it will still be filled with a background color or image. Margins, on the other hand, add white space (often called a gutter) between elements, giving the overall look of the page a lighter appearance.

You can control each side of the margin or padding for an element independently.
Four properties control margin edges: ***margin-top, margin-right, margin-bottom,and margin-left.***
Four properties control padding: ***padding-top, padding-right, padding-bottom, and padding-left.***
We can use any valid CSS measurement to define the size of a margin or padding, like so:

```
margin-right: 16px;
padding-top: 1em;
margin-left: 10%;
```

Margin and Padding Shorthand :

If we want to set all four sides of a style's margin or padding. But typing out all four properties for each style gets tedious, Instead we can use the shortcut properties named margin and padding to set all four properties quickly:

```
margin:     0      10px  10px  20px;
padding:    10px   5px   5px   10px;
```

The order in which you specify the four values is important. It must be **top, right, bottom, and left**.the easiest way to keep the order straight is to remember to stay out of **TR**ou**BL**e—top, right, bottom,and left.
If we want to use the same value for all four sides, it's even easier—just use a single value. If you want to remove margins from all <h1> tags, you can write this style:

```
h1 {  margin: 0; }
```

Similarly, use shorthand to add the same amount of space between some content and its border:

```
h1 { padding: 10px; }
```

Inline, Block, and Other Display Settings :

Web browsers treat every tag as a kind of box, not all boxes are alike.

CSS has two different types of boxes - **block boxes** and **inline boxes** that correspond to the two types of tags block-level and inline tags.

A block-level tag creates a *break* before and after it. The <p> tag, for example, creates a block that's separated from tags above and below. Headlines, <div> tags, tables, lists, and list items are other examples of block-level tags.

Inline tags don't create a break before or after them. They appear on the same line as the content and tags beside them. The <strong> tag is an inline tag. A word formatted with this tag happily sits next to other text—even text wrapped in other inline tags like <em>.

Border Property Shorthand

There are several border properties in CSS, But all these properties provide different ways of controlling the same three properties—color, width, and style—for each of the four borders. The most basic and straightforward property is border, which simply adds four borders:

```
border: 4px  solid  #FF0000;
```

The above style creates a solid, red, 4-pixel border. You can use this property to create a basic frame around a picture, navigation bar, or other item that you want to appear as a self-contained box.

The if we are writing the above shorthand property in longhand property mode in 3 lines of code :

```
border-width: 4px;
border-style: solid;
border-color: #FF0000;
```

We can also control each border individually using the appropriate property: ***border-top, border-bottom, border-left, or border-right.*** These properties work just like the regular border property, but they control just one side. The following property declaration adds a 2-pixel, green, dashed line below the style:

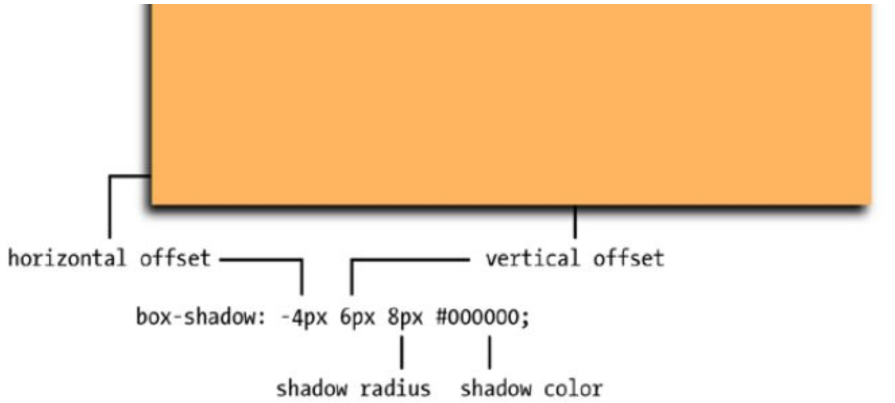
```
border-bottom: 2px  dashed  green;
```

<u>Adding Drop Shadows</u> We can add shadow effects to the text to make it pop from the page. CSS3 includes the *box-shadow property* to add drop shadows to an element's bounding box.

The syntax for the `box-shadow` property is as follows:

box-shadow: h-offset v-offset blur spread color inset ;

`h-offset` (required): A positive value moves the shadow to the right, while a negative value moves it to the left.
`v-offset` (required): A positive value moves the shadow downwards, while a negative value moves it upwards.
`blur` (optional): Specifies the blur radius of the shadow. A higher value creates a more blurred and spread-out shadow.
`spread` (optional): Specifies the spread radius of the shadow. It extends the size of the shadow.
`color` (optional): Specifies the color of the shadow.
`inset` (optional): If specified, the shadow appears inside the element. This creates an inner shadow effect.



```
box-shadow: -4px 6px 8px #000000;
```

Overview and features of CSS3

CSS3 is the latest version of CSS (Cascading Style Sheets), the language used for styling and presentation of web pages. It introduced a range of new features and enhancements to improve the capabilities and flexibility of CSS. Here is an overview of the key features of CSS3:

1. **Selectors:** CSS3 introduced advanced selectors such as attribute selectors, nth-child selectors, and more. These selectors allow for more precise and targeted styling of specific elements on a page.
2. **Box Model:** CSS3 added properties like box-sizing and border-radius to control the dimensions and appearance of boxes. This enables more flexible and responsive layouts.
3. **Transitions and Animations:** CSS3 includes properties like transition and animation that enable the creation of smooth transitions and animations without the need for JavaScript. This allows for engaging and interactive user experiences.
4. **Media Queries:** CSS3 introduced media queries, which allow styles to be applied based on different device characteristics such as screen size or resolution. This enables responsive design and optimal viewing experiences on various devices.
5. **Typography and Fonts:** CSS3 provides improved typography control with properties like @font-face, which allows custom fonts to be used, and properties like text-overflow and text-shadow for enhanced text effects.

6. **Flexbox and Grid Layout:** CSS3 introduced flexible box layout (Flexbox) and grid layout, which offer powerful and responsive methods for creating complex and flexible page layouts. These layout modules simplify the creation of dynamic and adaptive designs.
7. **Multiple Backgrounds:** CSS3 allows for multiple background images to be applied to an element. This provides more design possibilities and allows for layered background effects.
8. **Shadows and Effects:** CSS3 introduced properties like box-shadow and text-shadow for creating shadows and effects on elements. This enables the addition of depth and visual interest to page elements.
9. **Rounded Corners:** CSS3 introduced the border-radius property, allowing for the creation of elements with rounded corners. This adds a modern and visually appealing look to page elements.
10. **Gradients:** CSS3 allows for the creation of gradient backgrounds using the linear-gradient and radial-gradient functions. This provides smooth color transitions and enhances the visual aesthetics of elements.

CSS :Animation

INTRODUCTION

In HTML, animation refers to the process of creating dynamic and visually appealing effects that bring elements to life on a web page. HTML provides various methods and technologies to achieve animation effects, including CSS animations, JavaScript animations, and the use of external animation libraries and frameworks.

In other terms, **an animation is nothing more than a visualization of change.**

An animation is nothing more than a visualization of something changing over a period of time.

CSS Animations: CSS animations allow you to create animations using ***CSS properties and keyframes.*** With CSS animations, you can define the intermediate states of an element's properties over a specified duration. This allows for smooth transitions and transformations of elements without the need for JavaScript. CSS animations are controlled using CSS properties and can be triggered by adding CSS classes, applying pseudo-classes, or using JavaScript to modify the CSS properties dynamically.

Animation in HTML is a powerful tool to enhance user experience, engage visitors, and create visually appealing and interactive web pages. Whether using CSS animations, JavaScript animations, or external animation libraries, animation adds life and interactivity to static HTML content.

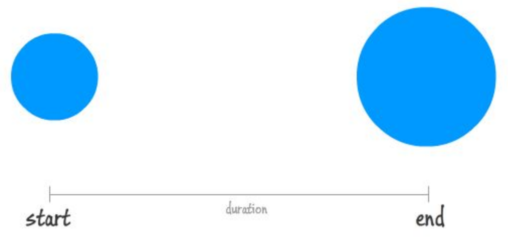
The Start and End States :

In CSS animation, the start and end states refer to the ***initial and final states of an element's properties*** that define the animation's behavior. These states determine how the element will appear and behave before and after the animation is applied.

Start State: The start state represents the initial appearance or behavior of the element before the animation begins. It is defined by the element's default CSS properties or any explicitly set CSS properties that are in effect before the animation starts. The start state serves as the foundation upon which the animation will be applied.

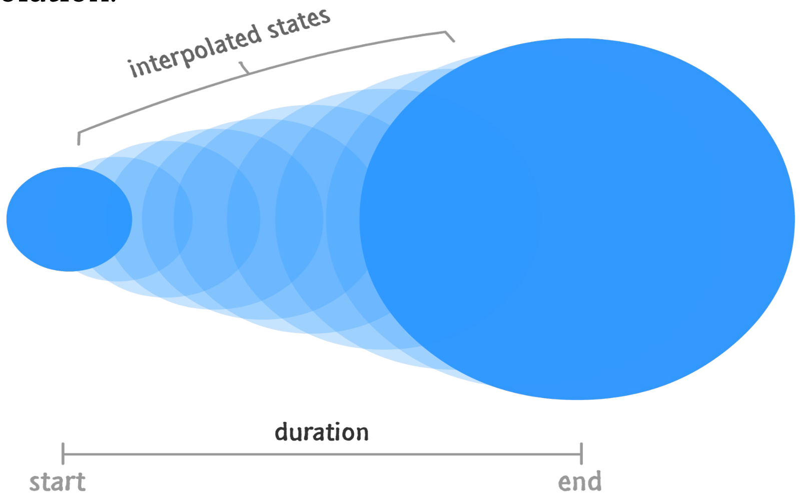
End State: The end state represents the desired appearance or behavior of the element when the animation is complete. It is defined by the properties specified in the final keyframe of the animation. The end state determines how the element will look or behave at the end of the animation.

By defining the start and end states, CSS animations provide the ability to transform and transition an element's properties, resulting in dynamic and visually appealing effects on web pages.



Interpolation

At the beginning, we have our start state and the end state. In order to make an animation out of what we have, we need a smooth transition that creates all the intermediate states between start and end states. This creation of the intermediate states is known as interpolation.



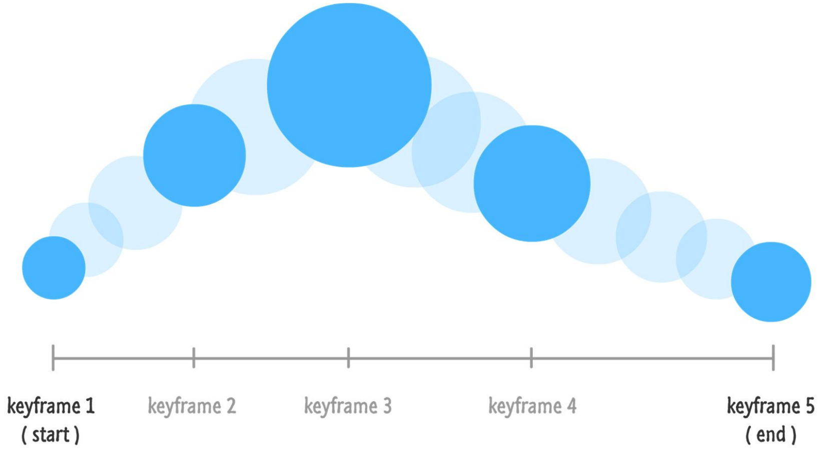
You may be wondering who specifies the interpolated states. The answer is that your browser or HTML rendering engine will take care of the messy details. All we need to specify is the **starting state**, the **ending state**, and the **duration** over which the transition between the two states needs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

Animations in HTML

In HTML, there isn't just a single animation approach that we can use. That would be too easy. Actually we have **three flavors of animations** to choose from, and each one is specialized for certain kinds of tasks.

1. CSS Animations (Keyframe Animations) :

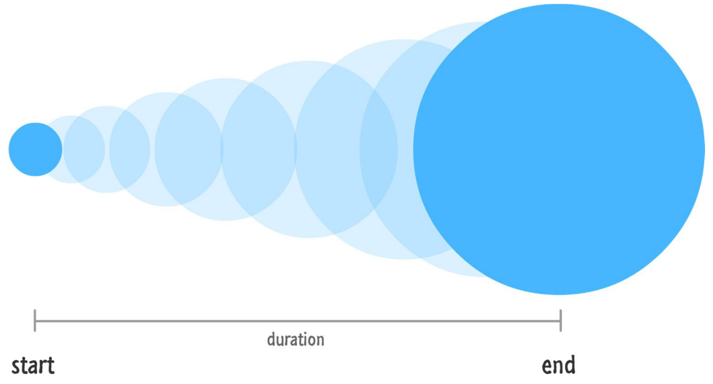
CSS Animations are the traditional animations that are on some sort of performance enhancing substance that makes them more awesome. With these kinds of animations, you can define not only the beginning and the end state but also any intermediate states commonly (and affectionately!) known as **keyframes**:



CSS animations are a popular and straightforward way to create animations in HTML. They leverage the power of CSS properties and keyframes to define the intermediate states of an element's properties over a specified duration. CSS animations offer smooth transitions and transformations without the need for JavaScript.

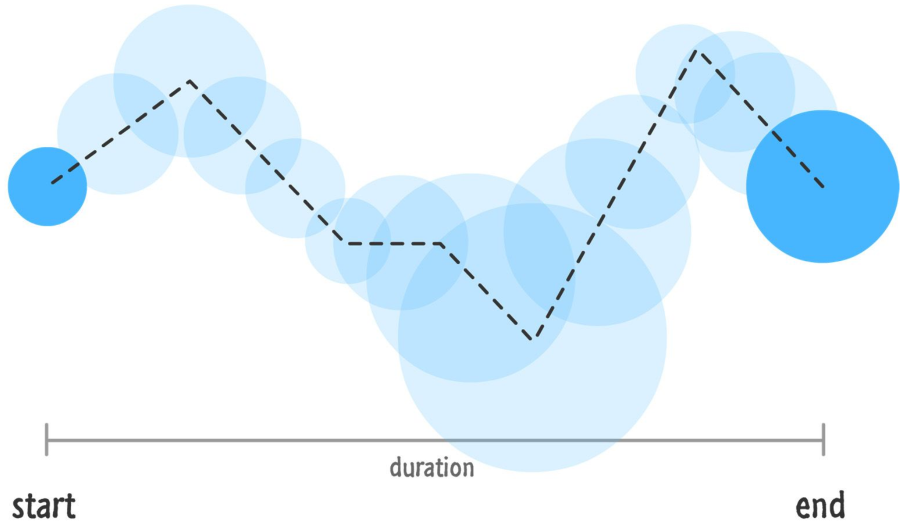
2. CSS Transitions :

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:



3. Javascript Animation :

If you want full control over what your animation does right down to how it interpolates between two states, you can use JavaScript. JavaScript provides more flexibility and control over animations compared to CSS animations. With JavaScript, you can programmatically manipulate an element's properties over time, creating dynamic and interactive animations. JavaScript animations are often used for complex animations, user interactions, or animations triggered by specific events.



All About CSS Animations – Creating a Simple Animation

What CSS animations do is pretty simple. They allow you to animate CSS properties on the elements they affect. This allows you to create all sorts of cool things like making things move, having things fade in and out, seeing things change color, etc.

CSS animations allow you to create dynamic and visually appealing effects on web pages. They involve defining keyframes that specify intermediate states of an element's properties over a duration. CSS animations are controlled using animation properties and can be triggered by adding CSS classes or using JavaScript.

Creating a Simple Animation:

To create a simple CSS animation, you need to define keyframes that specify the intermediate states of an element's properties over time. Here are the steps to create a basic animation :

I. Define the Keyframes
II. Apply the Animation
III. Trigger the Animation

I. Define the Keyframes:

Use the `@keyframes` rule to define the animation keyframes. It contains the @keyframes declaration followed by a name. Keyframes represent the different states of an element during the animation. On the inside, it contains style rules (aka the actual keyframes) whose selectors are either **percentage values** or the keywords **from and to**.

These keyframe style rules are pretty much what you would expect. They just contain CSS properties such as transform , opacity whose value will get applied when the rule becomes active.

II. Apply the Animation :

Apply the animation to an HTML element using the **animation** property. Specify the *animation name, duration, timing function, delay, and iteration count etc.*

III. Trigger the Animation :

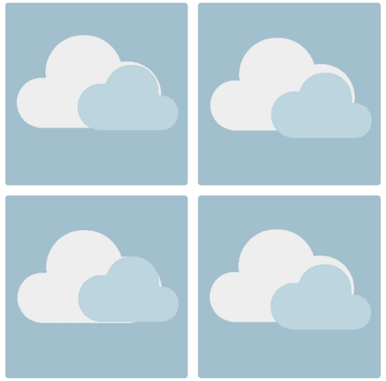
To start the animation, you can use various methods. For example, you can add the CSS class with the animation properties to the element, toggle the class with JavaScript, or trigger the animation on certain *events like hover or click*.

Let's look at an example , The animation we need to imagine is one where two clouds are bouncing up and down

```
#mainContent {
        background-color: #A2BFCE;
        border-radius: 4px;
        padding: 10px;
        width: 600px;
        height: 300px;
}
.cloud {
        position: absolute;
}
#bigcloud {
        margin-left: 100px;
        margin-top: 15px;
        animation-name: bobble;
        animation-duration: 2s;
        animation-iteration-count: infinite;
}
#smallcloud {
        margin-top: 65px;
        margin-left: 200px;
        animation-name: bobble;
        animation-duration: 4s;
        animation-iteration-count: infinite;
        }

@keyframes bobble {
        0% {
                transform: translate3d(50px, 40px, 0px);
                animation-timing-function: ease-in;
        }
        50% {
                transform: translate3d(50px, 50px, 0px);
                animation-timing-function: ease-out;
        }
        100% {
                transform: translate3d(50px, 40px, 0px);
                animation-timing-function: ease-in;
        }
}
```

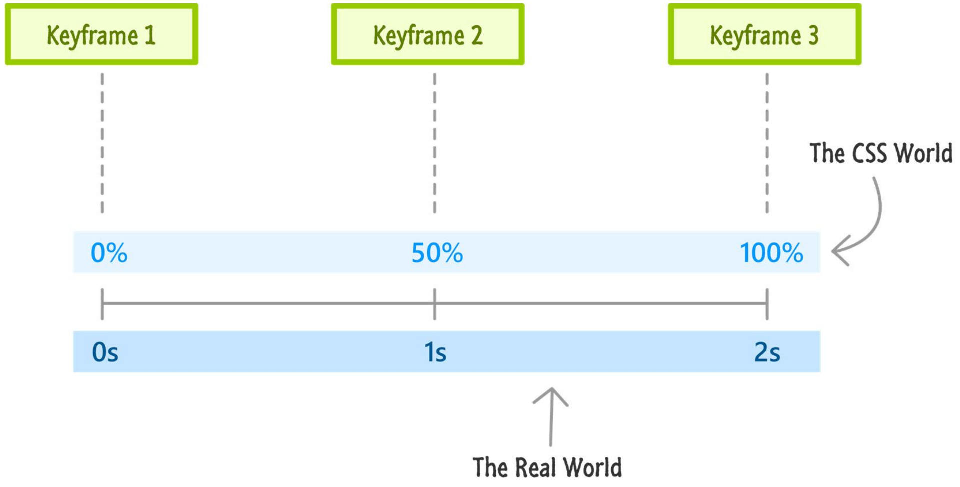
```
<!DOCTYPE html>
<html>
<head><title>Bouncing Clouds</title>
 <link rel=”stylesheet” href=”style.css”> </head><body>
<div id="mainContent">
<img id="bigcloud" alt="Big" class="cloud" height="154" src="bigCloud.webp"
width="238">
<img id="smallcloud" alt="Small" class="cloud" height="103"
src="smallCloud.webp" width="158">
</div>
</body>
</html>
```



In our example, the percentage values for our keyframe selectors are 0%, 50%, and 100%. What they represent is the percentage of the animation that has completed. When your animation is just starting, you have completed 0% of your animation. The 0% keyframe will become active. When your animation is half-way done, the 50% keyframe becomes active. At the end of your animation, the 100% keyframe becomes active.

The duration value you specify all the way over there on the animation property, besides setting the total time your animation runs, helps you to scale the values with opacity. You know, the way things are normally done.

Below is how our percentage values map to units of time for our 2 second animation:



Keyframes in Animation :

CSS Keyframes are used to define a series of intermediate states or keyframes for an animation. They specify how an element should appear at various points during the animation timeline. Keyframes are defined using the **@keyframes** rule in CSS.

The syntax for defining keyframes is as follows:

@keyframes animationName {

Define keyframe **selectors** { **properties** to change}

}

Here's a breakdown of the keyframes syntax:

animationName: This is a unique name given to the animation. It serves as a reference when applying the animation to an element using the animation-name property.

keyframe Selectors: Within the @keyframes rule, you define keyframe selectors that represent different points in time during the animation. Keyframe selectors can be specified using percentages or specific times. Commonly used percentages are **0% or from** (start of the animation) and **100% or to** (end of the animation).

properties: Inside each keyframe selector, you specify the CSS properties and values that you want to animate. These properties define how the element should appear at that specific point in the animation.

Here's an example that demonstrates the usage of keyframes:

```
@keyframes myAnimation {
        0% {
                opacity: 0;
                transform: scale(0.5);
        }
        50% {
                opacity: 1;
                transform: scale(1.2);
        }
        100% {
                opacity: 0;
                transform: scale(0.5);
        }
}
```

In this example, **myAnimation** is the name given to the animation. Three keyframe selectors are defined: **0%, 50%,** and **100%**. Each keyframe selector specifies the properties and values that should be applied at that particular point in the animation.

At the start of the animation (0%), the element has an opacity of 0 and is scaled to 0.5 of its original size. At 50%, the opacity becomes 1 and the element is scaled to 1.2 times its original size. Finally, at 100%, the opacity returns to 0 and the element is scaled back to 0.5.

By defining keyframes and their respective properties, you can create smooth and dynamic animations in CSS. The keyframes allow you to control the intermediate states of the animation, providing control over the element's appearance at different points in time.

Detailed Look at the CSS Animation Property :

The CSS animation property is a shorthand property that allows you to control various aspects of an animation applied to an element. It combines several individual animation-related properties into one, making it more convenient to defineand manage animations.

The syntax for the short-hand animation property is as follows:

animation: animationName duration timing-function delay iteration-count direction fill-mode play-state;

Let's take a detailed look at each component of the animation property:

1. animationName :

Specifies the name of the animation defined using **@keyframes**. You can provide multiple animation names separated by commas.If you want to apply multiple animations with different properties, durations, etc., We can define them separately and use the animation-name property instead of the shorthand.

2. animation-duration :

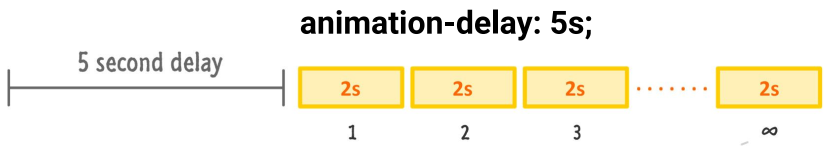
Sets the length of time that the animation takes to complete. You can specify the duration using seconds (s) or milliseconds (ms). Example **Values**: 2s (2 seconds), 500ms (500 milliseconds).

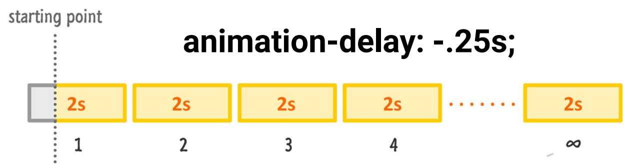
3.animation-timing-function :

Specifies the speed curve of the animation, determining how intermediate property values are calculated over time.
CSS provides several predefined timing functions, such as linear, ease, ease-in, ease-out, ease-in-out, and more.
We can also use custom timing functions created with cubic Bezier curves.
Example **Values**: linear, ease, ease-in, ease-out, ease-in-out, step-start, step-end cubic-bezier(0.25, 0.1, 0.25, 1).

4.animation-delay :

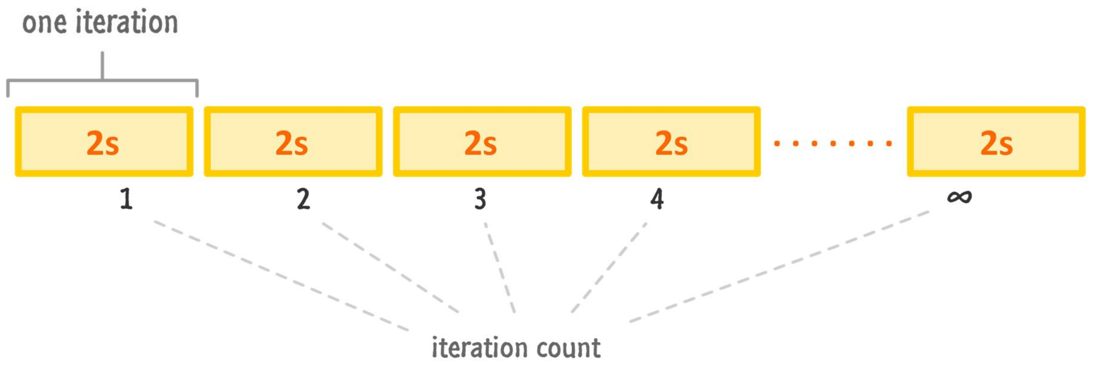
Sets a delay before the animation starts. You can specify the delay using seconds (s) or milliseconds (ms).
Examples **Values**: 5s (5 second), 200ms (200 milliseconds).
The delay occurs before the 0% keyframe from your first iteration is even hit: i.e: as below





5. animation-iteration-count :

Determines the number of times the animation should repeat. **Values** can be a positive integer, infinite, or alternate (which reverses the animation on each iteration). Examples: 3 (play animation 3 times), infinite, alternate.

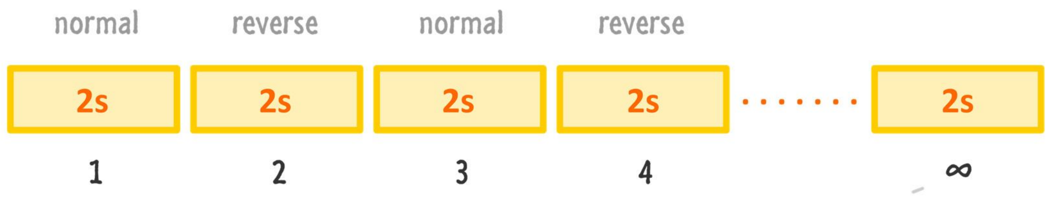


6. animation-direction :

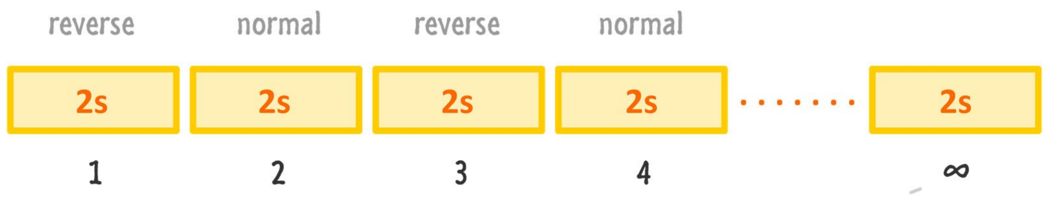
Defines whether the animation should play forward, in reverse, or alternate (forward and reverse)

Values can be normal (forward), reverse, alternate (forward and reverse on each iteration), or alternate-reverse (reverse and forward on each iteration).

When you set the **animation-direction** to **alternate-reverse**, your animation **starts off normal**. Starting with the second iteration, it plays in reverse and alternates between normal and reverse from there on out:



animation-direction to just **alternate** has a similar but slightly different behavior: animation **starts off in reverse** and alternates between normal and reverse from that point on.



7. animation-fill-mode :

Specifies how the element should be styled before and after the animation plays. I.e., If you want your starting keyframe's properties to apply during a delay or your last keyframe's properties to be retained after your animation has ended, you can set the animation-fill-mode property.

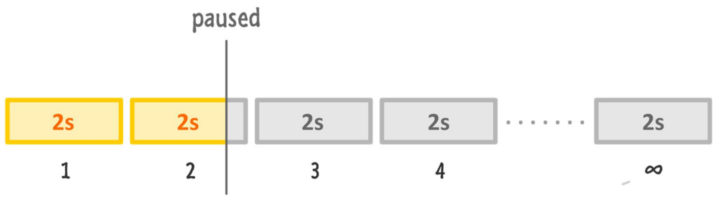
Values can be none (default, no styles are applied outside the animation),
forwards (retains the end state of the animation),
backwards (retains the start state of the animation),
both (retains both start and end states of the animation).

8. animation-play-state :

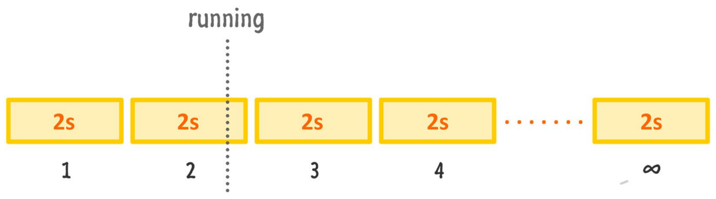
Determines whether the animation is **running** or **paused**.

This property allows you to toggle whether your animation plays or not by reacting to the running value or the paused value. By default, the animation-play-state property is set to running. We can set the value to paused to stop animation in its tracks:

animation-play-state : paused;



animation-play-state : running;



The Animation Shorthand :

The Longhand representation of the Animation Properties as follows :

```
#Someselector{
        animation-name: deathstar;
        animation-duration: 25s;
        animation-iteration-count: infinite;
        animation-play-state: paused;
        animation-delay: 0s;
        animation-direction: normal;
        animation-fill-mode: both;
        animation-timing-function: ease-out;
}
```

All of the longhand properties you see above can be represented in their shorthand form

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

So the above example can be represented as

i.e:
```
animation:    deathstar  25s  infinite   0s  normal  both  ease-out ;
animation-play-state: paused;
```

Declaring Multiple Animations :

As we have used single bouncing animation to two different clouds, Similarly we can declare different animations to the elements , which can be done in same **animation** property , using shorthand declaration or else using longhand declarations. In shorthand declaration, simply comma separates each of the animations as shown below:

```
#opStyle {
        animation: anim1 2s  infinite,  anim2 4s  infinite,  anim3  6s  3;
}
```

each animation is pointing to a different **@keyframes** rule. If for whatever reason you decide to point to the same @keyframes rule from within the same animation property declaration, based on CSS order precedence, the last one you declared will win.

When declaring the above animations in longhand, we have to do something that looks as follows:

```
#oppaGangnamStyle {
        animation-name: anim1 ,  anim2 ,  anim3 ;
        animation-duration: 2s, 4s,  6s;
        animation-iteration-count: infinite, infinite, 3;
}
```

CSS Transitions

CSS transitions provide a way to smoothly animate changes in CSS property values over a specified duration. They allow for gradual transitions between different property states, providing a smooth and visually appealing effect. Transitions can be applied to various CSS properties such as color, size, position, opacity, and more.

CSS transitions are an effective way to add subtle animations and transitions to web pages without the need for complex JavaScript code. They provide a simple and declarative approach to animating CSS properties, enhancing the user experience and adding visual interest to your designs.

“The transition property is a shorthand property used to represent up to four transition-related longhand properties [that] allow elements to change values over a specified duration, animating the property changes, rather than having them occur immediately.” - CSS Tricks

Transitions: Syntax

Shorthand :-

transition : property duration timing-function delay ;

Longhand :-

transition-property : property ;

transition-duration : duration ;

transition-timing-function : timing-function ;

transition-delay : delay ;

Adding a Transition :

To create a transition effect, you must specify at least two things:

- the CSS property you want to add an effect to
- the duration of the effect

The following example shows a 100px * 100px red <div> element. The <div> element has also specified a transition effect for the width property, with a duration of 2 seconds:

```
div {
     width: 100px;
     height: 100px;
     background: red;
     transition: width 2s;
}
```

The transition effect will start when the specified CSS property (width) changes value.

Now, let us specify a new value for the width property when a user mouses over the <div > element:

```
div:hover {
     width: 300px;
}
```

Notice that when the cursor mouses out of the element, it will gradually change back to its original style.

Looking at Transitions in Detail :

Let's take a detailed look at each component of the transition syntax:

1. transition-property:

Specifies the CSS property that will transition between different values. You can specify multiple properties separated by commas.

Examples: color, width, opacity.

2. transition-duration:

Sets the length of time the transition effect takes to complete. You can specify the duration using seconds (s) or milliseconds (ms).

Examples: 1s (1 second), 500ms (500 milliseconds).

3. transition-timing-function:

Determines the speed curve or timing of the transition. CSS provides several predefined timing functions such as linear, ease, ease-in, ease-out, ease-in-out, and more. You can also use custom timing functions created with cubic Bezier curves.

Examples: linear, ease-in-out, cubic-bezier(0.25, 0.1, 0.25, 1).

4. transition-delay:

Specifies a delay before the transition effect starts. You can specify the delay using seconds (s) or milliseconds (ms).

Examples: 0.5s (0.5 seconds), 200ms (200 milliseconds).

<u>Here's an example that showcases the usage of CSS transitions:</u>

```
.element {
  transition : width   1s   ease-in-out   0.5s,   background-color   0.5s   linear;
}
```

In this example:

- The transition is applied to the `.element` class.
- The `width` property will transition over a duration of `1s`, using an ease-in-out timing function, with a delay of `0.5s`.
- The `background-color` property will transition over a duration of `0.5s`, using a linear timing function, with no delay.

Transition : Longhand Properties vs Shorthand Properties

CSS transitions can be specified using either longhand properties or shorthand properties. Both approaches allow you to define the transition effect, but they differ in terms of syntax and flexibility.

Longhand properties allow you to specify individual aspects of the transition effect separately. The following longhand properties are used for transitions:

```
.element {
        transition-property : width, height ;
        transition-duration : 1s ;
        transition-timing-function : ease-in-out ;
        transition-delay : 0.5s ;
}
```

Longhand properties offer more flexibility as they allow you to control each aspect of the transition independently. This can be useful when you want to apply different transition settings to different properties.

Shorthand properties provide a more concise way to specify the transition effect. They combine multiple aspects of the transition into a single property. The shorthand property for transitions is **`transition`**.

The **`transition`** property accepts values in the following order:

- **`transition-property`**
- **`transition-duration`**
- **`transition-timing-function`**
- **`transition-delay`**

Example using shorthand property:

```
.element {
        transition : width  1s  ease-in-out  0.5s,  height  0.5s  linear;
}
```

Shorthand properties are more convenient when you have consistent transition settings across multiple properties. They provide a compact way to define the transition effect without repeating individual property declarations.

It's important to note that the shorthand property will override any individual longhand properties defined separately. If both are used, the shorthand property takes precedence.

Working with Multiple Transitions :

Multiple transitions can be applied to an element to create different effects on different CSS properties. By specifying multiple transition declarations, you can control how each property animates independently.

To apply multiple transitions, you can either use separate `transition` or longhand properties for each property, or you can combine them within a single `transition` or longhand property declaration.

Here's an example using separate `transition` declarations:

```
.element {
        transition-property : width ;
        transition-duration : 1s ;
        transition-timing-function : ease-in-out ;
        transition-delay : 0.5s ;

        transition-property : color ;
        transition-duration : 0.5s ;
        transition-timing-function : linear ;
        transition-delay : 0 ;
}
```

Alternatively, you can combine multiple transitions within a single declaration:

```
.element {

       transition : width  1s  ease-in-out  0.5s, color  0.5s  linear  0 ;

}
```

The timing-function property can have the following values:

- `ease` - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
- `linear` - specifies a transition effect with the same speed from start to end
- `ease-in` - specifies a transition effect with a slow start
- `ease-out` - specifies a transition effect with a slow end
- `ease-in-out` - specifies a transition effect with a slow start and end
- `cubic-bezier(n,n,n,n)` - lets you define your own values in a cubic-bezier function