6. CSS Animation

What are CSS Animations?

An animation lets an element gradually change from one style to another. You can change as many CSS properties you want, as many times you want. To use CSS animation, you must first specify some keyframes for the animation. Keyframes hold what styles the element will have at certain times.

CSS transitions

CSS transitions are one of the ways we can create animation with CSS, even though they are not called animations. By definition, animation means causing change over time and transitions absolutely do that. They just do it in a little bit of a different way than CSS keyframe animations do.

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.

How to use CSS Transitions?

To create a transition effect, we must specify two things:

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

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

Understanding CSS animation

CSS transform property

The `transform` property applies a 2D or 3D transformation to an element. This property allows you to rotate, scale, move, skew, etc., elements.

Transform property has values as:

translate(*x*,*y*)	Defines a 2D translation
translateX(*x*)	Defines a translation, using only the value for the X-axis
translateY(*y*)	Defines a translation, using only the value for the Y-axis
scale(*x*,*y*)	Defines a 2D scale transformation
rotate(*angle*)	Defines a 2D rotation, the angle is specified in the parameter

For any CSS keyframe animation we create, we have to first define the animation. Basically, tell CSS what it is that should happen and we also need to assign that animation to a specific element or elements in your HTML. We can do these two steps in any order.

@keyframes

We will start creating our CSS animation by defining it's keyframes using the **`@keyframes`** rule. **`@keyframes`** are essentially a list describing everything that should happen over the course of the animation. We define the values for the animating properties at various points during the animation and any property that we will see change over the course of one's cycle of an animation.

- The **`@keyframes`** rule works like most other CSS rules, anything contained within its curly braces are considered part of that block.

- The second step to our **`@keyframes`** rule is giving our animation a name. It's really important to name the animation because without a name, it would not be able to assign the animation to an element.

There are a few options available to us for how we can define each **`@keyframes`** within our **`@keyframes`** rule:

- **`from, to`**: You define where to animate from and where to animate to.

Activity) create an animation that will move an image from one place to another within the x-axis.

For this exercise, we will download two images, one for the moving object and the other one as a back image. Once we have the images, will be create two containers for each of the images using `<div>` with position relative and absolute respectively.

HTML

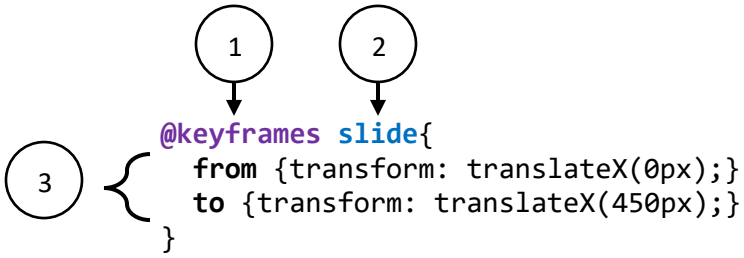
```
<h3>CSS animation using @keyframes with attributes from - to </h3>
<div class="mountainImage"><img src="mountain.jpg"/>
    <div class="busImage"><img src="bus.png"/></div>
</div>
```

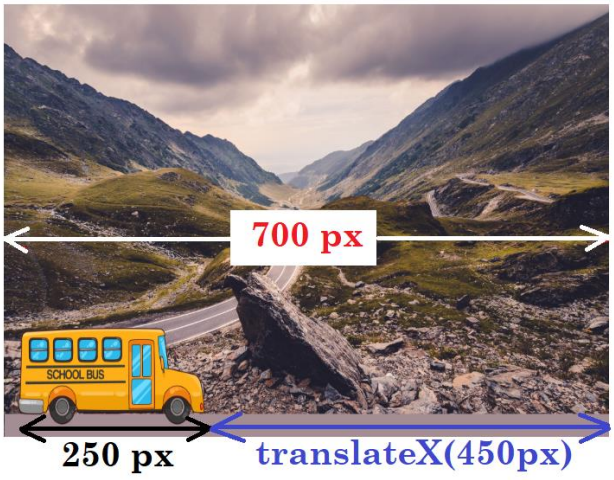
CSS

```
.mountainImage{
  width: 700px;
  height: 500px;
  position: relative;
  margin: auto;}
.busImage{
  width: 250px;
  height: 150px;
  position: absolute;
  bottom: 0;}
```

Once we have the images set up, we can start writing the css animation using @keyframes. We are going to name this animation *slide*. The animation will make the element to move from left to right. The basic rules to use @keyframes are:

1. Use @keyframes to create the animation frame
2. Assign a name to the keyframe
3. Specify what do you want the animation to do. For this project, we want the image to move *from* left *to* right. For this, we need to transform the element within the X axis from position 0 to 220px.





Now that we have the animation created, we need to add it into the image that we want to be animated, in our case, we will add the animation to the element that has the bus image, which has the class name ***busImage***:

```
div class="busImage"><img src="bus.png"/></div>
```

Now in css, we call for the class name and add some animation properties to it:

1. **`animation-name: slide;`** ➔ This tells our image with class name **`busImage`** to apply the **`@keyframes`** animation named **`slide`**

2. **`animation-duration: 2s;`** ➔ The second property we need to define is the animation duration. Our keyframes define what should change over the course of an animation, but they don't give any indication as to how long this should take, so that's what we do in animation duration. For this case, our animation **`slide`** is set to 2 seconds.

```
.busImage{
    animation-name: slide;
    animation-duration: 3s;
}
```

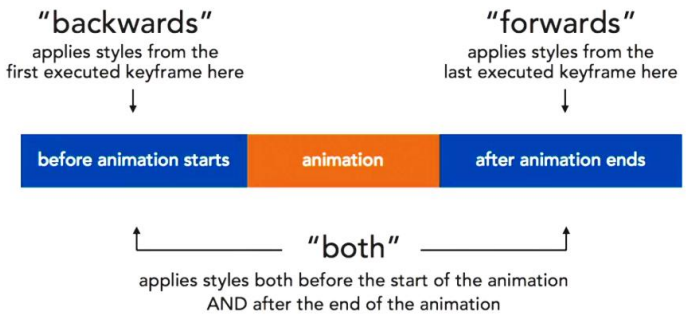
3. The **`animation-iteration-count`** property is also a good one to set for yourself even though it has a default value. This one determines how many times the animation will play. By default, the animation runs once, but we can set it runs **`infinite`** times.

```
animation-iteration-count: 2;
```

4. **`animation-delay`** property sets the waiting before the animation actually executes.

```
animation-delay: 2s;
```

5. **`animation-fill-mode`** is a way of telling the animating element what to do outside of the actual duration of its animation and it can take four values: *none, backwards, forward, and both*. The default value is *none*. If we use the value *forward*, the image will stay at the position at the end of the animation.




```
animation-fill-mode: forwards;
```

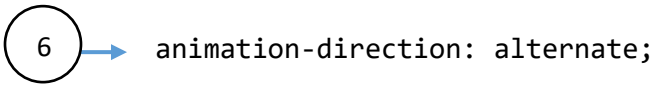
6. **animation-direction** let's us manipulate what order our **keyframes** are executed in. Animation-direction can be set to *normal, reverse, alternate, and alternate-reverse.*

   An animation-direction of **normal** means that all iterations of the animation will be played as specified. So your keyframes will play from start to end. That means your keyframes will play from your from keyframe to your to keyframe, or your 0% percent keyframe to your 100% keyframe exactly in the order that you wrote them.

   The animation direction of **reverse** means that all iterations of your animation will be played in the reverse direction. Your last keyframe will be played first, and your first keyframe will be played last. So this would play your animation from your to keyframe to your from keyframe, or your 100% keyframe to your 0% keyframe.

   An animation-direction of **alternate** means that the direction of your animation will alternate each iteration of the animation. It will play in the normal direction the first time. So that would be 0% keyframe to 100% keyframe, and in the reverse direction the second time. So that would be the 100% keyframe back to 0%. **alternate** can only be used if your animation plays more than once because otherwise you won't have any space to see the alternating. So we'll change our animation iteration count to two. With those two small changes made, we'll go back to our robot and see how it's changed.

   An **alternate-reverse** works in very much the same way as alternate, but it starts playing your animation in the reverse direction first, and then normal. **alternate-reverse** works in much the same way as *alternate,* it plays in reverse the first time, and then forwards, but it still plays twice.



animation-timing-function is the way speed is distributed across the duration of our animation. The in animation-timing-function keywords include *ease, linear, ease-in, ease-out, and ease-in-out*.

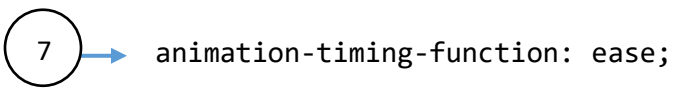
With the **linear** easing our robot moves across the screen at the same constant speed for the entire animation. Linear easing creates a constant speed of motion that never changes at all. This is often perceived as mechanical motion because nothing in real life actually moves like that.

ease allows the animation to start and slowly and then speed up to a uniform speed.

ease-in has a distinct starting slow and then speeding up as the image reaches it's destination.

ease-out is the opposite of ease-in. The image starts at a higher speed and then slows down as it gets to it's destination.

ease-in-out combines the effects of ease-in and ease-out. It starts out a little bit slow, hits a top speed in the middle of the animation, and then slows down as it reaches the end.



Infinitely looping animation

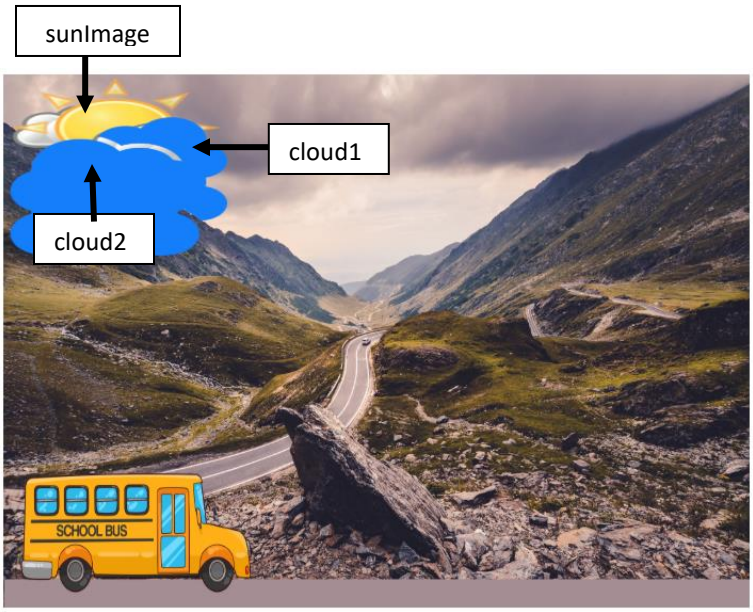
CSS is great for looping animation because it makes it possible to set an infinite loop with much less code than other methods.

Example) Using the previous exercises, creating a couple of animated clouds that continently drift across the sky with different distance and speed. For this, we can download a cloud image and create three containers of different sizes for each cloud.

HTML

```
<div class="mountainImage"><img src="mountain.jpg"/>
      <div class="busImage"><img src="bus.png"></div>
      <div class="sunImage"><img src="sun.png"></div>
      <div class="cloud1"><img src="cloud1.png"></div>
      <div class="cloud2"><img src="cloud1.png"></div>
</div>
```

Since the clouds are displayed on top of the back image, the containers' position must be set to absolute. We can also set different width and height to the clouds container and also the top position to each cloud.



CSS

```
/* animation clouds */
.sunImage{
  width: 30%;
  height: 20%;
  position: absolute;
  top: 0;
}
.cloud1{
  width: 30%;
  height: 26%;
  position: absolute;
  top: 20px;
}
.cloud2{
  width: 30%;
  height: 26%;
  position: absolute;
  top: 60px;
}
```

Once we have the HTML and CSS set, now we can set the animation using the @keyframes. Since we want the clouds to flow from left to right, we can use the property **`from`** to **`to`**

CSS

```
@keyframes cloudsFlowing {
  from {transform: translateX(0);}
  to {transform: translateX(500px);}
}
```

We are going to use the same animation for all three clouds, but we are going to adjust the properties to make each cloud to behave in a slightly different way. For example, we can set different **animation-iteration-count** to each cloud.

css

```
. sunImage{
  width: 30%;
  height: 20%;
  position: absolute;
  top: 0;
  animation-name: cloudsFlowing;
  animation-duration: 30s;
  animation-timing-function: linear;
  animation-iteration-count: infinite;
}
```

The animation properties can also be written in one line using the property **animation**. For example, we can simplify the following properties:

```
animation-name: cloudsFlowing;
animation-duration: 30s;
animation-timing-function: linear;
animation-iteration-count: infinite;
```

using **animation** property, we can write each property values in one line and separate the values with a space:

```
animation: cloudsFlowing 30s linear infinite;
```

Now, we can apply the **cloudsFlowing** animation to the other two clouds with 8 and 5 seconds respectively.

css

```
.cloud1{
  width: 30%;
  height: 26%;
  position: absolute;
  top: 20px;
  animation: cloudsFlowing 10s linear infinite;
}
.cloud2{
  width: 30%;
  height: 26%;
  position: absolute;
  top: 60px;
  animation: cloudsFlowing 6s linear infinite;
}
```

Simplify the animation lines using **animation** property

opacity

One of the transition properties is **opacity**. **Opacity** property happens when a new value is assigned to it. It will cause a smooth change between the old value and the new value over a period of time. The value of opacity is ranged in between 0 and 1 where 0 is clear and 1 is the full color of the element.

Activity) Using the previous animation, make the clouds to be almost half visible when they reach the edge of the right side.

CSS

```
@keyframes cloudsFlowing {
  from{transform: translateX(0); opacity: 0.9;}
  to{transform: translateX(500px); opacity: 0.3;}
}
```

CSS animation using "%" time frame

CSS animation in an element can be slip using % time frame. For example, from a **@keyframes** animation using **from** to **to** attributes, we can replace **from** to **0%** and **to** to **100%**:

```
@keyframes cloudsFlowing{
  from{ transform: translateX(0px); opacity:1;}
  to{transform: translateX(250px);opacity:0.5;}
}

@keyframes cloudsFlowing{
  0%{ transform: translateX(0px); opacity:1;}
  100%{transform: translateX(250px);opacity:0.5;}
}
```