

Chapter 6

CSS Animation

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.

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 its 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) In this activity, we will create an animation that will move an image from one place to another within the x-axis.

First, prepare a cordova project folder (chapter 1). Open a text editor and save a HTML file as *index.html* in the www folder within the cordova project folder.

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, we will create two containers for each of the images using `<div>` with position relative and absolute respectively.

```
<h3>CSS animation using @keyframes with attributes from - to </h3>
<div class="mountainImage"></div>
<div class="busImage"></div>
```

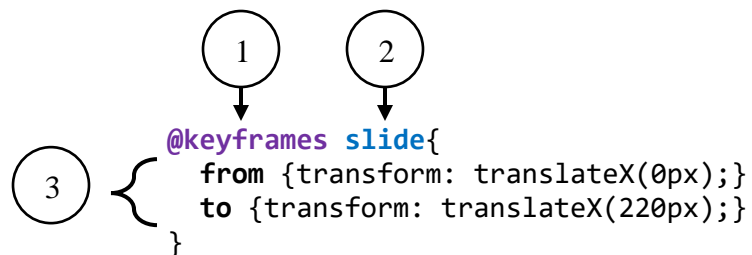
HTML

```
.mountainImage{
  width: 100%;
  height: 380px;
  position: relative;
}
.busImage{
  width: 30%;
  height: 80px;
  position: absolute;
  bottom: 0px;
}
```

CSS

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.

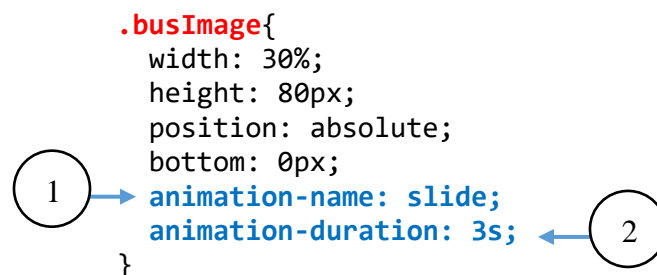


Now that we have the animation created, we need to add it into the image that we want to be animated, in our case, we will add the animation to the element that has the bus image, which has the class name *busImage*:

```
div class="busImage"></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.



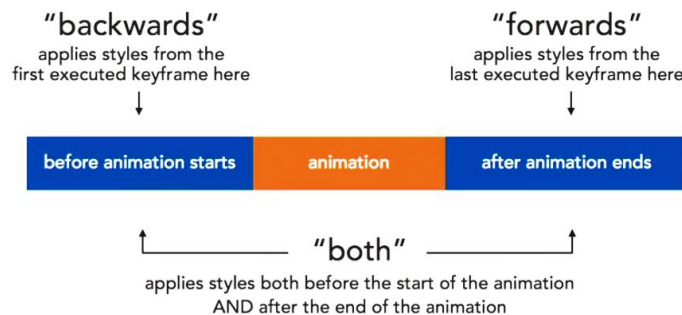
3. The **animation-iteration-count** property is also a good one to set for yourself even though it has a default value. This one determines how many times the animation will play. By default, the animation runs once, but we can set it runs *infinite* times.

3 → `animation-iteration-count: 2;`

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

4 → `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.



5 → `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.

6 → `animation-direction: alternate;`

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.

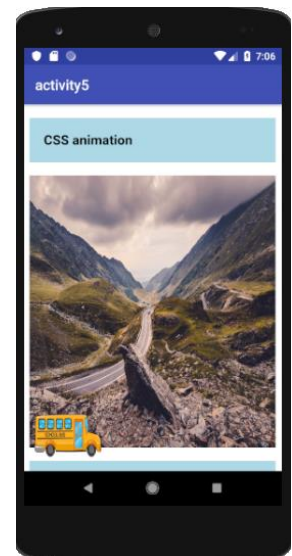
ease allows the animation to start and slowly and then speed up to a uniform speed.

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.

7 → `animation-timing-function: ease;`



Infinately 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.

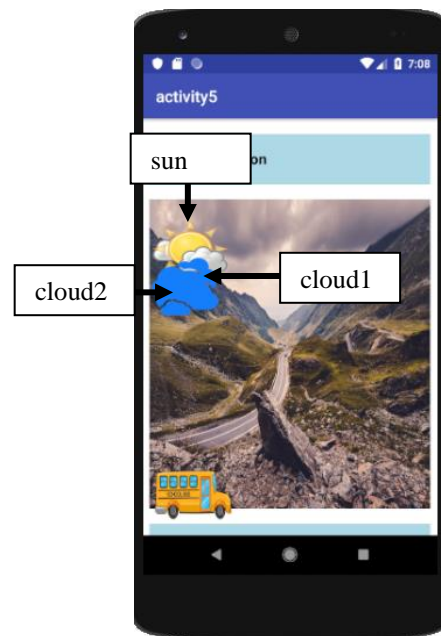
```
<div class="sun"></div>
<div class="cloud1"></div>
<div class="cloud2"></div>
```

HTML

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.

```
/* animation clouds */
.sun{
  width: 100px;
  height: 80px;
  top:0 px;
  position: absolute;
}
.cloud1{
  width: 80px;
  height: 70px;
  top: 50px;
  position: absolute;
}
.cloud2{
  width: 90px;
  height: 70px;
  top: 120px;
  position: absolute;
}
```

CSS



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**

```
@keyframes cloudsFlowing {
  from {transform: translateX(0);}
  to {transform: translateX(300px);}
}
```

CSS

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.

```
.sun{
  width: 100px;
  height: 80px;
  top:120px;
  position: absolute;
  animation-name: cloudsFlowing;
  animation-duration: 30s;
  animation-timing-function: linear;
  animation-iteration-count: infinite;
}
```

CSS

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.

```
.cloud1{
  width: 80px;
  height: 70px;
  top: 160px;
  position: absolute;
  animation: cloudsFlowing 8s linear infinite;
}
.cloud2{
  width: 90px;
  height: 70px;
  top: 180px;
  position: absolute;
  animation: cloudsFlowing 5s linear infinite;
}
```

CSS

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

CSS animation using “%” time frame

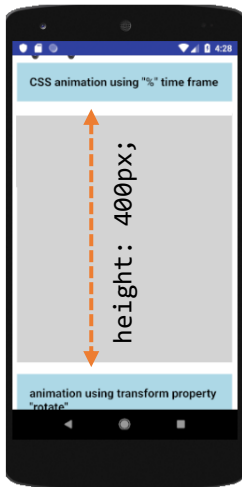
CSS animation in an element can be slip using % time frame. For example, from a **@keyframes** animation using **from** to **to** attributes, we can replace **from** to **0%** and **to** to **100%**:

```
@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;}
}
```


Activity) Create an animation that will move a square from one position to another and change its background color at the different position. The total time of the animation is 10s and the animation will repeat 20 times.

To understand how the position works on an element, first we need to create a container to place the square:



HTML

```
<h3>CSS animation using \"%\" time frame </h3>
<div class="frameContainer">
</div>
```

CSS

```
.frameContainer{
  height: 400px;
  background-color: lightgray;
  position: relative;
}
```

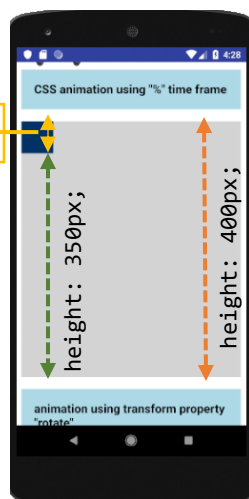
Once we have the container for the square, we create the square that sits inside the container:

Add this line to create a square using `<div>` element

```
<div class="frameContainer">
  <div class="animationSquare"></div>
</div>
```

HTML

height: 50px;



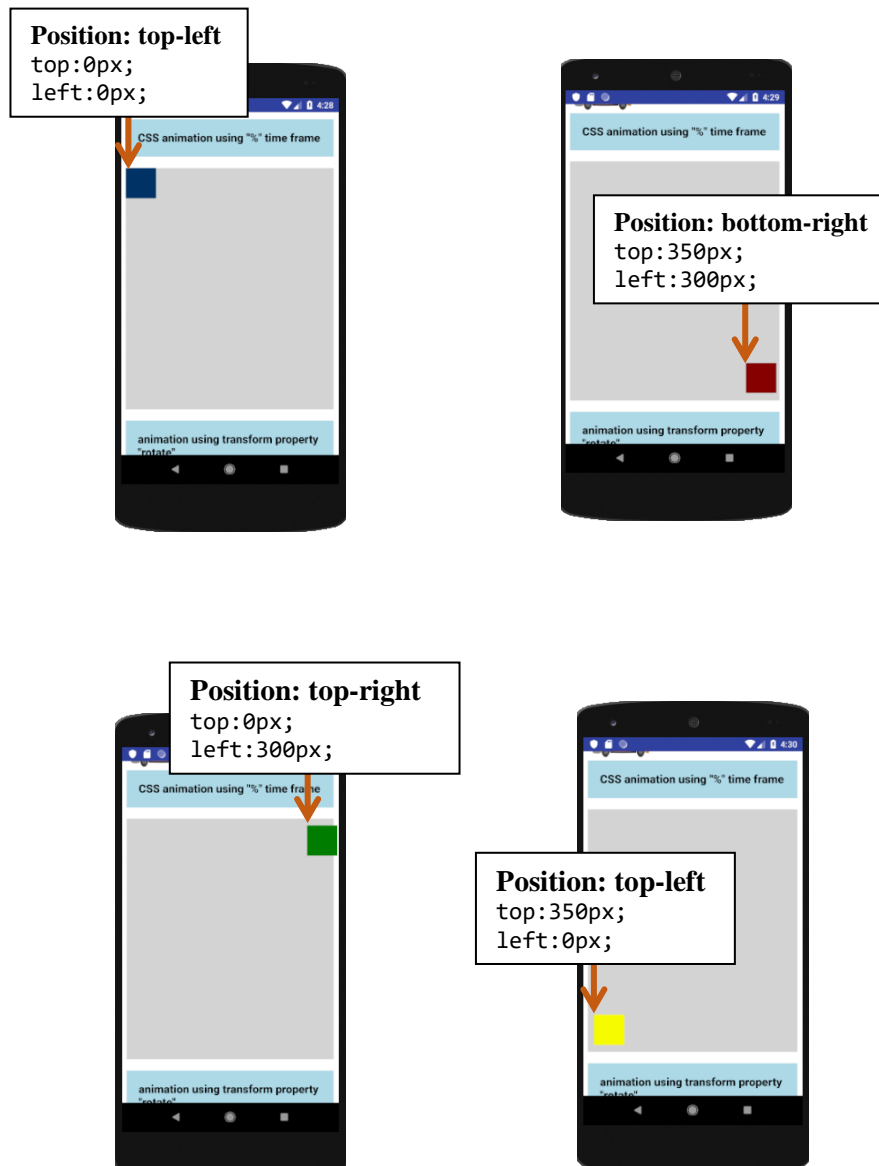
CSS

```
.animationSquare{
  width: 50px;
  height: 50px;
  background-color: rgb(10,50,100);
  position: absolute;
  animation: animation_square 10s 20;
}
```

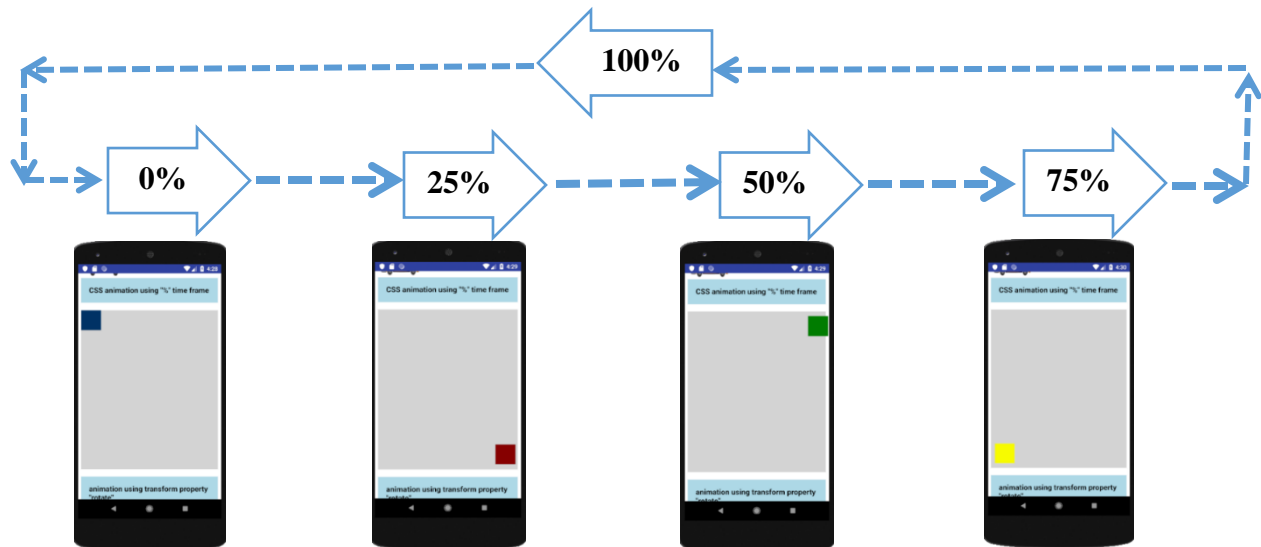
After the square is set, we need to create the animation using % time frame as the following:

Total time	10 seconds				
Time frame	0%	25%	50%	75%	100%
	0 to 2 secs	2.1 to 4 secs	4.1 to 6 secs	6.1 to 8 secs	8.1 to 10 secs
Background color	Darkblue	Darkred	Green	Yellow	darkblue
Position	Top-left	Bottom-right	Top-right	Bottom-left	Top-left

To create the animation, we need to understand the position first. The position of an element is measured from the top-left corner of the element:



One complete cycle of the animation:



CSS

```
@keyframes animation_square{
  0%   {background-color:darkblue; left:0px; top:0px;}
  25%  {background-color:darkred; left:300px; top:350px;}
  50%  {background-color:green; left:300px; top:0px;}
  75%  {background-color:yellow; left:0px; top:350px;}
  100% {background-color:darkblue; left:0px; top:0px;}
}
```

Rotating animation

Rotation is a CSS property that defines a 2D rotation and the rotation angle is specified within the parenthesis.

For example, if we want to rotate an element 200 degree clockwise, we can write: `rotate(200deg)`

If we want to rotate an element 200 degree counter-clockwise, we can write: `rotate(-200deg)`

Activity) Create an animation that will rotate an image clockwise 180 degree and then 360 degree counter-clockwise. The animation duration will be 6s and it will repeat 10 times.

For this animation, first, we create the animation structure in HTML and CSS:

HTML

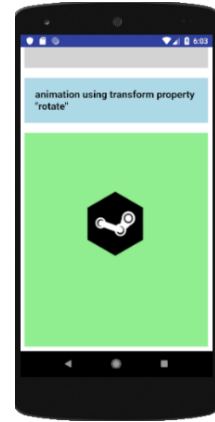
```
<h3>animation using transform property "rotate"</h3>
<div class="frameRotation">
  <div class="rotationImage">
    
  </div>
</div>
```

```

.frameRotation{
  width: 100%;
  height: 400px;
  background-color: lightgreen;
  position: relative;
}
.rotationImage{
  width: 35%;
  height: 30%;
  padding: 110px;
  animation: spin linear 6s 10;
  position: absolute;
}

```

CSS



Once the structure is set, we can start the animation according to the following time frame:

Total time	6 seconds			
Time frame	0%	30%	70%	100%
rotation	Rotate 0 deg	Rotate 180 deg	Rotate -360 deg	Rotate 0 deg

```

@keyframes spin {
  0%{transform: rotate(0deg);}
  30%{transform: rotate(180deg);}
  70%{transform: rotate(-360deg);}
  100%{transform: rotate(0deg);}
}

```

CSS

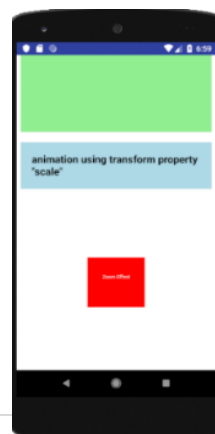
Zooming an image

CSS animation transform property has value **scale()** to create zoom-in and zoom-out effect on an element. There are different parameters that we can use within the **scale**'s parenthesis, for now, we are going to use one value inside the parenthesis. The highest value is 1 and the lowest value is 0. 1 means 100% of the actual element's size, and 0 means 0% of the actual element's size.

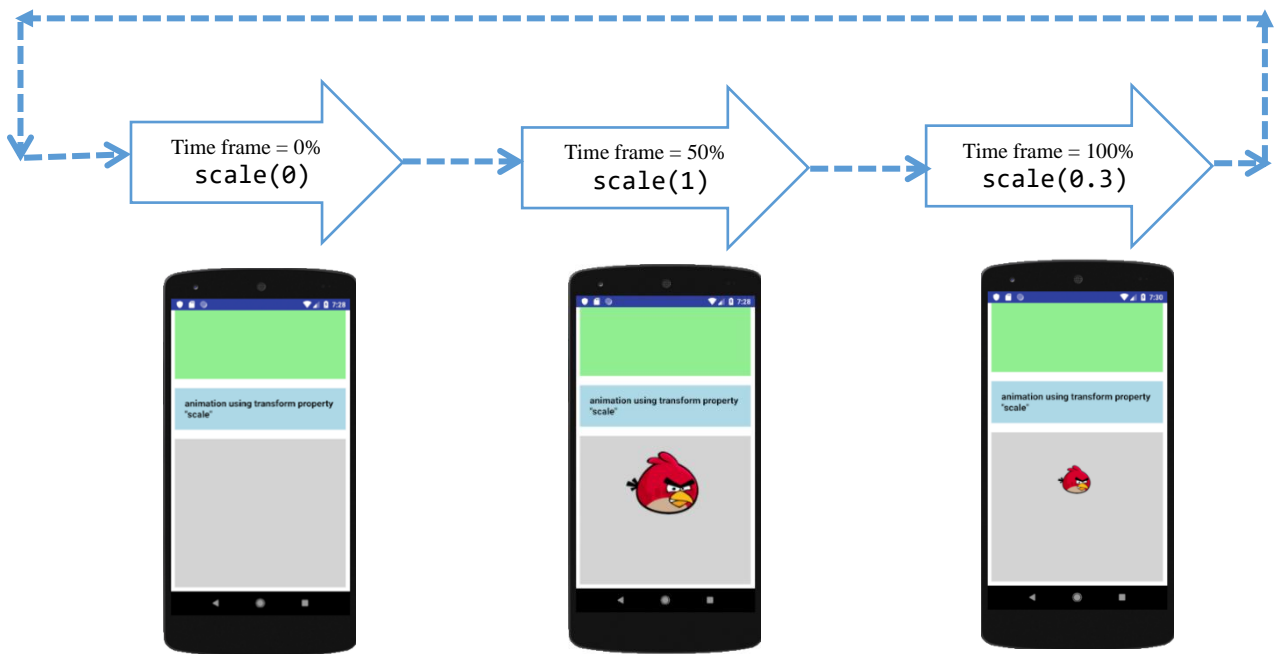
100% of the element size
transform: scale(1);



30% of the actual size
transform: scale(0.3);



Activity) Create an animation that the element will zoom-in to 100% of the image and zoom-out to 30% of the image. The animation duration is 10s and it will play 5 times.



HTML

```
<h3>animation using transform property "scale"</h3>
  <div class="scaleAnimation">
    <div class="zoomImage"></div>
  </div>
```

CSS

```
.scaleAnimation{
  width: 100%;
  height: 400px;
  background-color: lightgreen;
  position: relative;
}
.zoomImage{
  width: 50%;
  height: 50%;
  position: absolute;
  animation: zoomAnimation 10s 5;
}
@keyframes zoomAnimation {
  0%{transform: scale(0); top: 50px; left: 30px;}
  50%{transform: scale(1); top: 50px; left: 30px}
  100%{transform: scale(0.3); top: 50px; left: 30px}
}
```

Combining Animation

IMPORTANT NOTE

The materials used in this manual have the author's rights and are for educational use only.