P.PORTO

POLITÉCNICO
DO PORTO
ESMAD

COMPUTAÇÃO GRÁFICA
TSIW

Syllabus

- CSS Transitions
- CSS Animations

SVG animations

- **SMIL** (*Synchronized Multimedia Integration Language*) **animations**: animation functionalities for XML documents (like SVG)
  - It is, however, being slowly deprecated

```
<svg width="800" height="500" >
      <circle cx="200" cy="200" r="100" stroke="black" stroke-width="5" fill="red">
            <animate attributeName="cx" from="200" to="400"
            begin="0s" dur="5s" repeatCount="indefinite" />
      </circle>
</svg>
```

https://codepen.io/teresaterroso/pen/mdyjdWp

Animations API

- **Web Animations API**: new standard, to provide access to the animation engine of the browsers, allowing more complex and fluid animations
  - It aims to bring the power of CSS performance, add the benefits and flexibility of JavaScript
  - Still in a **draft stage**

```
<div id='photo'>
   <img src= 'photo.jpg' />
</div>

<script>
        document.getElementById("photo").animate(  [
                { transform: 'translate3D(0%, 0%, 0)', backgroundColor: '#000'  },
                { transform: 'translate3D(10%, 10%, 0)', backgroundColor: '#ff0000' }
        ], { duration: 3000, iterations: Infinity});
</script>
```

https://codepen.io/teresaterroso/pen/PowBoOa

CSS animation

- **CSS animation**: allows for animations described in CSS language, to animate CSS properties

- **Externals libraries**: while outside the scope of this course, a more curious reader can be pointed to several libraries that offer a variety of animation methods for UI designers

GreenSock (GSAP)	Animate.css
VelocityJS	Bounce.js
BonsaiJS	Anime.js
VivusJS	Magic Animations
RaphaelJS	Zdog
SVG.js	CSShake
Snap.svg	Hover.css
Lazy Line Painter	Walkway

CSS animation

- Provides animation to almost all HTML elements, without the need for JavaScript
  - As it would be expected, SVG falls on the category of a HTML element, and thus it can be animated using CSS animation
- CSS animation describes **how the animatable CSS properties** change over time
  - `fill`, `stroke`, `background-position`, `transform`, … are CSS properties whose values can change over time
- The animation can either be triggered by a state transition (e.g., the user hovers an element), or it can be an explicit property of an element

CSS animation: transitions

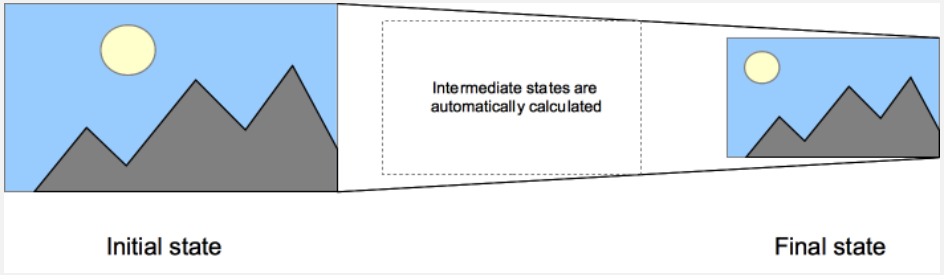
- CSS transitions: **definition of CSS styles that are used on specific conditions**
- Those conditions are defined by **pseudo-classes**, and specify a special state of an element
- Examples of pseudo-classes (and thus states) are:
  - `:hover` - selects links on mouse over
  - `:active` - selects the active link
  - `:focus` - selects the <input> element that has focus
- CSS syntax for pseudo-classes:

```
selector.pseudo-class {
        property: value;
}
```

  - For a complete list of pseudo-classes, visit this link

CSS animation: transitions

- It is typical to **create CSS styles for when a special state occurs**
  - For example, when the mouse hovers a link, a feedback is usually given
- With CSS, when a state transition is triggered, it is possible to instruct the browser to perform a **smooth transition between the properties**, instead of just changing them, as it normally does

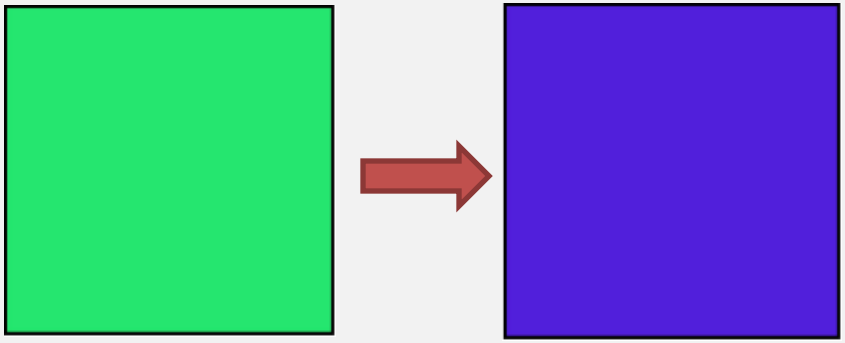


- For a transition to take place, an element must have a **change in state**, and a **different property value** must be declared on that state change

CSS animation: transitions

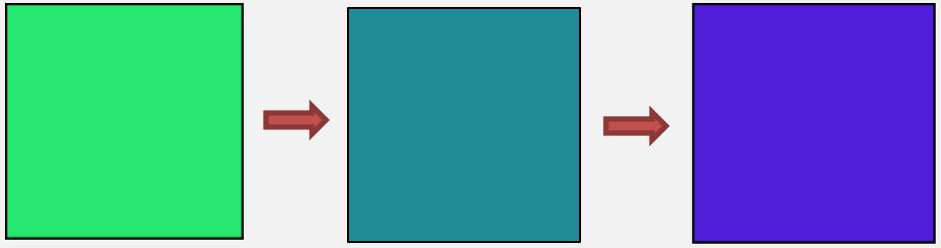
- **Immediate** state change applied to a SVG square

```
svg rect {
        fill: #00E969 ;
}
svg rect:hover {
        fill: #5100DF ;
}
```



- **Smooth** state change applied to a SVG square (using **CSS transitions**)

```
svg rect {
        fill: #00E969 ;
        transition: all 2s;
}
svg rect:hover {
        fill: #5100DF ;
}
```



watch animation

CSS animation: transitions

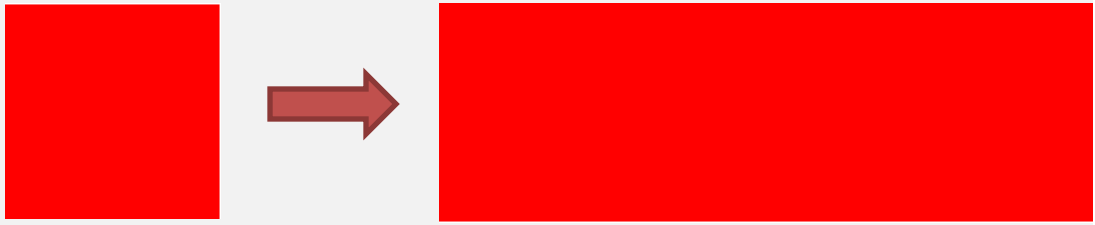
- Another example

HTML

```
<div class="demo"></div>
```

CSS

```
.demo {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}
.demo:hover {
  width: 300px;
}
```



watch animation

CSS animation: transitions

- To create an animated transition between CSS animatable properties, use the **`transition` property**
- This property is a **shortcut** for several sub-properties:
  - `transition-property`: property, or properties, to animate; the special keyword `all` represents all properties at the same time
  - `transition-duration`: (mandatory) time in seconds or milliseconds that the animation takes from start to finish; no duration means no transition (default: 0s)
  - `transition-timing-function`: specifies the speed curve for the transition, and can take different values (default: ease – see next slide)
  - `transition-delay`: time in seconds (s) or milliseconds (ms) before the transition animation starts (default: 0s)

CSS timing functions

- `transition-timing-function`: specifies the speed curve for the transition, and can take the values:
  - `linear`: the same speed from start to end
  - `ease`: (**default**) transition effect with a slow start, then fast, then end slowly
  - `ease-in`: starts slowly, and accelerates gradually until the end
  - `ease-out`: starts quickly, and decelerates gradually until the end
  - `ease-in-out`: transition effect with a slow start and end (like ease, but more pronounced)
  - `cubic-Bezier(n,n,n,n)`: define the values in a cubic-bezier function

watch animation

CSS animation: transitions

- Not all the four transition-related properties are required to build a transition
- CSS transitions can be controlled using only the shorthand `transition` property or by explicitly declare its sub-properties

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

```
#delay {
        font-size: 40px;
        transition-property: font-size;
        transition-duration: 4s;
        transition-delay: 2s;
}

#delay:hover {
        font-size: 60px;
}
```

```
#delay {
        font-size: 40px;
        transition: font-size 4s 2s;
}

#delay:hover {
        font-size: 60px;
}
```

watch animation

CSS animation: transitions

- More than one property can be animated with transitions

```
#heart {
        transform-origin: 25% 25%;
        fill: red;
        transition: fill 1s, transform 4s;
}

#heart:hover {
        transform: scale(1.5);
        fill: green;
}
```

```
#heart {
        transform-origin: 25% 25%;
        fill: red;
        transition-property: fill, transform;
        transition-duration: 1s, 4s;
}

#heart:hover {
        transform: scale(1.5);
        fill: green;
}
```

watch animation

CSS animation: transform

- With the CSS `transform` property one can perform the following 2D or 3D transformation methods on HTML elements:
  - `translate(x,y)` / `translateX(x)` / `translateY(y)`: 2D translations
  - `translate3d(x,y,z)` / `translateZ(z)`: 3D translations
  - `scale(x,y)` / `scaleX(x)` / `scaleY(y)`: 2D scaling
  - `scale3d(x,y,z)` / `scaleZ(z)`: 3D scaling
  - `rotate(θdeg)`: 2D rotation
  - `rotateX(θdeg)` / `rotateY(θdeg) )` / `rotateZ(θdeg)` : 3D rotation
  - `skewX(θdeg)` / `skewY(θdeg)`: 2D skews
  - `initial`: sets the property to its default value

CSS transforms playground

CSS animation: transform

- While trivial to apply, some transform (namely **scaling** and **rotation**) may not perform as expected
- The reason is that the position of the **transform origin** is by default the top left corner
- Luckily, CSS provides a simple property to change the origin:
  `transform-origin: x-axis y-axis z-axis`
  - Accepted values for each axis are

    ```
    x-axis: left | center | right | length | %
    y-axis: top | center | bottom | length | %
    z-axis: length
    ```
  - E.g., `transform-origin: 50% 50%` places the origin at the center of the element

Watch animation

CSS animation: transform

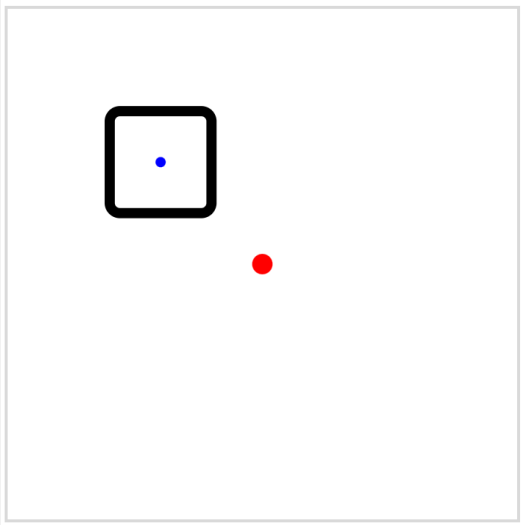
- The `transform-box` CSS property may also be important in rotating and scaling of HTML objects
  - It defines the layout box to which the `transform-origin` property relates

```
svg { … }
#box {
        /* the same as transform-origin: center center;*/
        transform-origin: 50% 50%;

       /* uncomment this and the square will rotate around the red circle
          – the SVG element center*/
        transform-box: fill-box; /* the square rotates around the blue circle, its center*/

       transform: rotate(0deg);
}

svg:hover #box {
        transform: rotate(360deg);
        transition: transform 1s;
}
```



Watch animation

CSS animation: transitions & JS

- Transitions are a great tool to make things look much smoother without having to do anything to your JavaScript functionality
  - With JS one can make the animation happens and with CSS it can be turned smoother without any extra effort

HTML

```
<div class="ball"></div>
```

CSS

```
.ball {
  border-radius: 25px;
  width: 50px;
  height: 50px;
  background: #c00;
  position: absolute;
  transition: transform 1s;
}
```

JavaScript

```
let ball = document.querySelector(".ball");
document.addEventListener(
  "click",
  function (event) {
    ball.style.transform  = "translateY(" + (event.clientY - 25) + "px)";
    ball.style.transform  += "translateX(" + (event.clientX - 25) + "px)";
  },
  false
);
```

watch animation

CSS animation: transition events

- Several events are fired during CSS transitions, that can be used to run some JavaScript functions based on the current state of a transition
  - One may want to know when an animation starts before beginning a timer, or when a transition ends, some some analytic data could be sent
  - **`transitionstart`**: runs as soon as the actual transition starts, and after any `transition-delay` value has elapsed
  - **`transitionend`**: runs when a transition ends in both directions: on completion and also when reverting back to the initial state

    This event will not run if:

    1. The element being transitioned is removed from the DOM
    2. The transition property is removed
    3. The element is set to `display:none` at some point during the transition
  - **`transitionrun`**: runs when a transition is created, but before any `transition-delay` begins
  - **`transitioncancel`**: runs if a transition is cancelled between `transitionrun` and before `transitionend`

Watch animation

CSS animation: transition events

- Adding and removing CSS classes makes light work of applying CSS transitions on demand
- With the **`classList`** API, one can store CSS transitions to be played in a CSS class, and play and reverse them on demand by adding or removing them to an element
  - It represents the contents of an HTML element's class attribute
  - This list can be modified using `add()`, `remove()`, `replace()` and `toggle()` methods
  - https://developer.mozilla.org/en-US/docs/Web/API/Element/classList
  - https://developer.mozilla.org/en-US/docs/Web/API/DOMTokenList/toggle#examples

CSS animation: transition events

- Example of adding/removing a transition by click on the HTML element

HTML

```
<div class="demo"></div>
```

CSS

```
/* initial class */
.demo {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}
/* animate class added on click */
.demo.animate {
  width: 300px;
}
```

JavaScript

```
let item = document.querySelector(".demo");
item.addEventListener("click", (e) => {
  // adds a new class
  item.classList.toggle("animate");
  item.addEventListener("transitionend", transitionEndCallback);
});

transitionEndCallback = (event) => {
  console.log(`transition of property ${event.propertyName} has
ended`);
  item.removeEventListener("transitionend",
transitionEndCallback);
  item.classList.remove("animate"); // removes class
};
```

watch animation

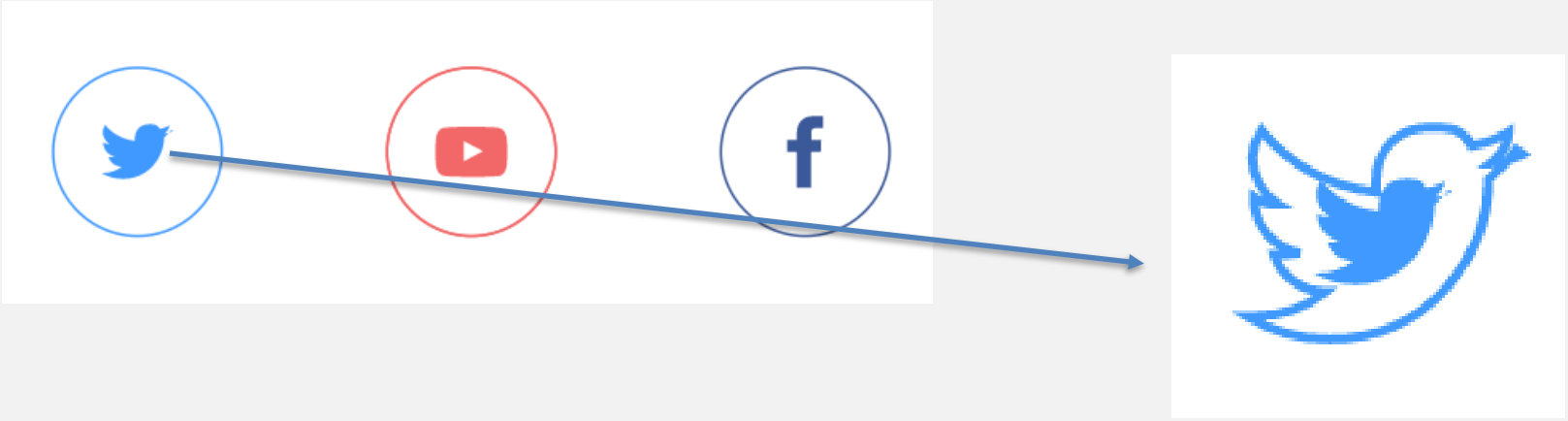
Try yourself….

1. Use the base file available in Moodle, which contains an HTML page with a SVG element with two circles, like the image below. Using CSS transitions, when user hovers the mouse cursor over the SVG element, move the pink circle to the right and then stop it; then, move the small blue circle to the right when “hited” by the large pink circle
   HINT: use **translations** to move them

Try yourself….

2. Use the base file available in Moodle. Consider the following well-knowed icons, in SVG. Use CSS transitions to animated the hover state (duration for all: 1s).
   - Twitter icon group: when the user hovers the mouse cursor over, make the outline scale down to 0, and a second icon appear by changing the stroke color from transparent to #4099ff, and at the same time scale it up to 2

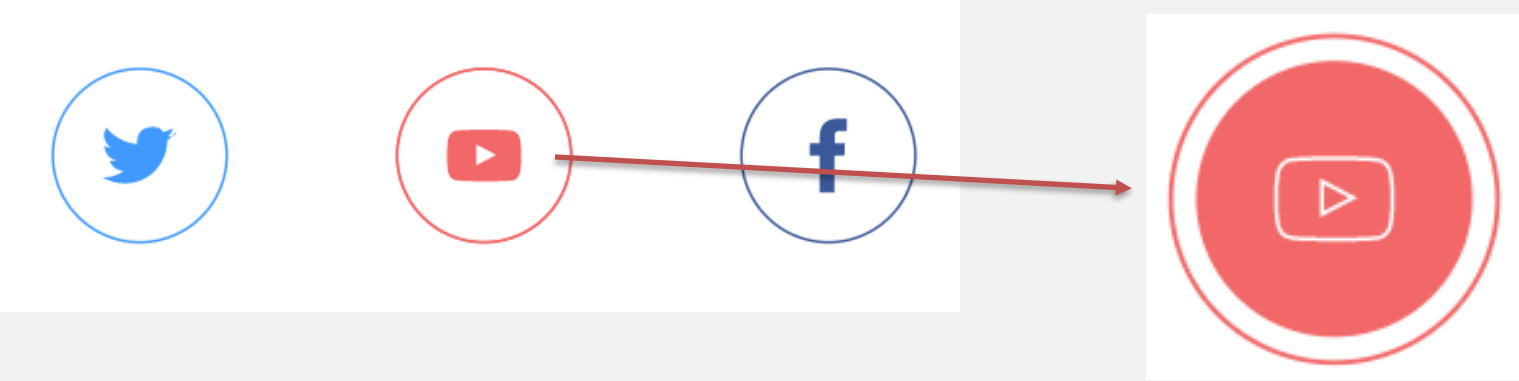
HINT: use the following origin for both scale transformations (22% 50%)



Try yourself….

2. Use the base file available in Moodle. Consider the following well-knowed icons, in SVG. Use CSS transitions to animated the hover state (duration for all: 1s).
   - Youtube icon group: when the user hovers the mouse cursor over, make the outline scale up to 1.2, change the inner circle fill color to #f26768 and the icon stroke color to white

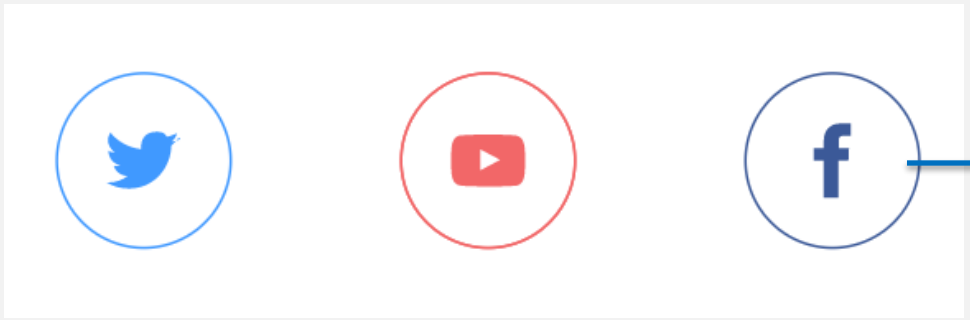
HINT: use the following origin for the scale transformation (50% 50%)



Try yourself….

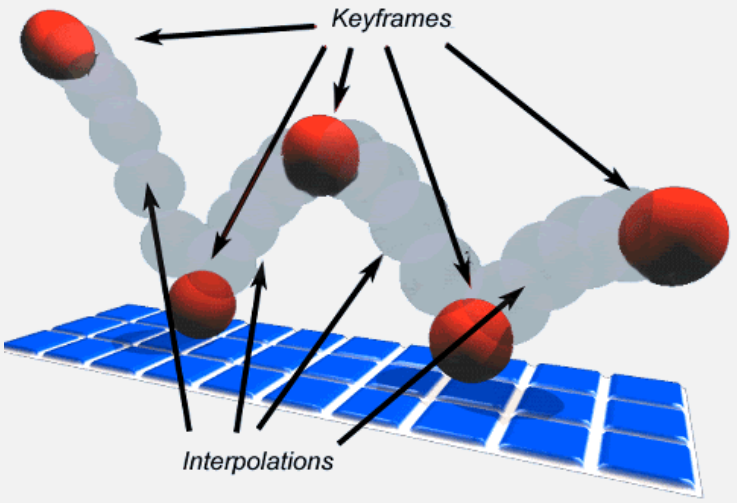
2. Use the base file available in Moodle. Consider the following well-knowed icons, in SVG. Use CSS transitions to animated the hover state (duration for all: 1s).
   - Facebook icon group: when the user hovers the mouse cursor over, make the outline circle disappear by change its opacity to 0 and at the same time scale it up to 1.8, make the detail appear and at the same time scale it down to 0.8

HINT: use the following origin for both scale transformations (78% 50%)



CSS animation: animations

- But what to do when one just wants to animate one or more elements, **without a state transition**?
- The CSS standard specifies one **property** - `animation` - and one **rule** - `@keyframes`, that, when used together, produce **animation**
- **Keyframing** is an animation technique in which special frames are marked, and the properties of an object are explicitly given
  - For the remaining frames, the values in between the keyframes are **interpolated**
  - This gives more control over the intermediate steps of the animation sequence than transitions



CSS animation: animations

- CSS rule `@keyframes` syntax:

```
@keyframes animation_name {
    keyframes-selector {
        css-styles;
    }
}
```

animation_name: name by which this particular animation will be referred

keyframes-selector: `from`, `to` or percentage along the animation (0%-100%)

css-styles: values of the properties at those places

CSS animation: animations

- The following keyframed animation that, once applied will change the fill color (of whatever element it is applied to)

```
@keyframes square-animation {
        0% {
                fill: #00E969;
        }
        100% {
                fill: #5100DF;
        }
}
```

```
@keyframes square-animation {
        from {
                fill: #00E969;
        }
        to {
                fill: #5100DF;
        }
}
```

CSS animation: animations

- This 2nd example shows a more complex animation, named `scaleit`, there are three keyframes:

```
@keyframes scaleit {
        0% {
                fill: red;
        }
        80% {
                transform: scale(1);
                fill: green;
        }
        100% {
                transform: scale(2);
                fill: green;
        }
}
```

CSS animation: animations

- Defining keyframes animations, per-se, does not animate anything
- The actual animation occurs when the **`animation`** property of CSS is used

HTML

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis commodo, ipsum ac molestie pulvinar, diam ante egestas odio, nec venenatis est urna in nisi. In eget leo est. Nunc gravida euismod urna. Sed semper dictum neque porta sollicitudin. Morbi porta laoreet ultrices. Praesent eleifend libero ac tellus sodales eleifend. In id justo porttitor, ultrices lorem vitae, malesuada lorem.</p>
```

watch animation

CSS

```
p {
  animation: 3s slidein;
}

@keyframes slidein {
  from {
    margin-left: 100%;
    width: 400%;
  }
  to {
    margin-left: 0%;
    width: 100%;
  }
}
```

CSS animation property

- Like `transition`, **`animation`** is a shortcut for a number of properties:
  - `animation-duration`: (mandatory) time in seconds or milliseconds that the animation takes from start to finish; no duration means no transition
  - `animation-timing-function`: speed curve for the animation (same values as in transitions)
  - `animation-delay`: time in seconds (s) or milliseconds (ms) before the animation starts
  - `animation-iteration-count`: number of times an animation should be played:

    number | `infinite`

CSS animation property

- **`animation`** sub-properties:
  - `(…)`
  - `animation-direction`: whether an animation should be played forwards, backwards or in alternate cycles. Alternate supposes repetition:

    `normal` | `reverse` | `alternate` | `alternate-reverse`
  - `animation-fill-mode`: element style when the animation is not playing (before it starts, after it ends, or both), or in which CSS style is the animation going to end:

    `none` | `forwards` | `backwards` | `both`
  - `animation-play-state`: whether the animation is running or paused:

    `paused` | `running`

CSS animations: animation

- MDN: Using CSS animation
- If one wants to play the `scaleit` animation for 2 seconds (duration), by an infinite number of times (iteration count) and alternating (played forwards, then backwards, and so on...), one writes:

```
@keyframes scaleit {
        0% {
                transform: scale(1);
                fill: red;
        }
        80% {
                transform: scale(1);
                fill: green;
        }
        100% {
                transform: scale(2);
                fill: green;
        }
}

element {
        animation: 2s scaleit alternate infinite;
}
```

watch animation

CSS animation events

- Like CSS transition events, it is possible to get additional control over animations — as well as useful information about them — by making use of animation events
  - These events can be used to detect when animations start, finish, and begin a new iteration
  - Each event includes the time (in seconds) at which it occurred as well as the name of the animation that triggered the event

watch animation

Try yourself….

1. Implement the following keyframe animations for the 4 circles on the file available in Moodle: they all move horizontally 400 pixels, during 5s.
   Change the necessary animation sub-properties, so that:
   a) Repeat the animation 2 times and, **at the end**, the 1st **circle returns to its original position after the animation**
   b) Repeat the animation 2 times and, **at the en**d, the 2nd **circle remains on the position set by the last keyframe**
   c) 3rd circle: the animation **never stops**
   d) 4th circle: the animation **never stops, but it is played forwards** first and then backwards

Try yourself….

2. Implement a linear keyframe animation for the circle on the file available in Moodle, with a duration of 5s, and never stops repeating.
   During the 5s of the animation the following properties should change:
   - stroke color: changes from black to green in the 1st half of the animation
   - stroke width: changes from 1 to 100
   - circle position: the circle translates horizontally from position 50px to position 450px (in the 1st half of the animation) and returns back to position 50px (in the 2nd half of the animation)

Try yourself….

3. Implement a linear keyframe animation for the path on the file available in Moodle, with a duration of 5s, and never stops repeating.
   Knowing that the path length is about 1000 pixels, animate is drawing by combining setting the `stroke-dasharray` SVG atribute or CSS property to 1000 and by animating the `stroke-dashoffset` CSS property.
   The animation must draw the path back and forward.

Try yourself….

4. Let's flip some cards:
   a) SVG group element `rim` is animated by the animation named `rectangle`, that scales up from 0 to 1 the red element, and at the same time, makes it from totally transparent to totally opaque
   duration: 1s delay: 5s

Try yourself….

4. Implement two keyframe animations on the file available in Moodle:
   b) SVG group element `text` is animated by the animation named `text-anim`, that animates CSS properties stroke-dashoffset, opacity and fill. In the beginning: stroke-dashoffset 210, totally transparent, no fill color. At 50%, stroke-dashoffset changes to 0, fill color #fafafa opaque. At 75%, fill color #F7931E and in the end, a 2D rotation transformation of 720 degrees. duration: 5s