SUBJECT: CSS Animations DATE:

CSS Transitions

- transitions get used to describe a property + how its changes should be animated

```css
.animated {
  transition-property: background-color;
  transition-duration: 3s;
}
```

↑ animates changes of background-color for 3s

```html
<button id="color">Click me</button>

<style>
  #color {
    transition-property: background-color;
    transition-duration: 3s;
  }
</style>

<script>
  color.onclick = function() {
    this.style.backgroundColor = 'red';
  };
</script>
```

Click me

→ clicking the button would animate the background for 3 seconds

- 4 properties to describe CSS Transitions:
  - transition-property
  - transition-duration
  - transition-timing-function
  - transition-delay

SUBJECT: CSS Transitions DATE:

✳ The single transition property can declare all 4 transition properties in a specific order

⟶ transition: property duration timing-function delay

```
<button id="growing">Click me</button>

<style>
#growing {
  transition: font-size 3s, color 2s;
}
</style>

<script>
growing.onclick = function() {
  this.style.fontSize = '36px';
  this.style.color = 'red';
};
</script>
```

Click me

- transition-property

⟶ can write a list of properties to animate

- Ex: left, margin-left, height, color, etc.

⟶ can write all to animate all properties

- Most of the commonly-used properties are animatable

- transition-duration

⟶ How long an animation should take in seconds (s) or milliseconds (ms)

SUBJECT: CSS Transitions DATE:

- transition-delay

→ specifies the delay <u>before</u> the animation
  - A transition-delay of 1s would start the animation 1 second after the property change

→ negative values are possible
  - the animation is shown immediately, but the starting point of the animation will be after the given value (time)

Example:

Result script.js style.css index.html

Click below to animate:

0

animation shifts num from 0-9 once clicked using translate

```
#stripe.animate {
  transform: translate(-90%);
  transition-property: transform;
  transition-duration: 9s;
}
```

```
stripe.classList.add('animate');
```

↳ adding the .animate class to the element will start the animation

can use transition-delay to start the animation at an exact num ↴

```
stripe.onclick = function() {
  let sec = new Date().getSeconds() % 10;
  // for instance, -3s here starts the animation from the 3rd second
  stripe.style.transitionDelay = '-' + sec + 's';
  stripe.classList.add('animate');
};
```

SUBJECT: transition-timing-function DATE:

transition-timing-function

- Determines if animation starts slowly then goes fast, or vice-versa
- this property accepts 2 value types:
  1. Bezier Curve (used more often)
  2. Steps

the timing function set as a Bezier Curve

- Is set w/4 control points:
  1. first control point → (0,0) fixed value
  2. last control point → (1,1) fixed value
  3. intermediate control points → x: 0..1
     y: anything
- Syntax → cubic-bezier(x2, y2, x3, y3)

(no need to specify the fixed values)

SUBJECT: transition-timing-function DATE:

- The timing function describes how fast the animation process goes

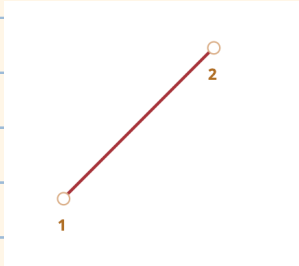
  → X-axis is the time of the transition-duration
    - 0 is the start
    - 1 is the end

  → y-axis specifies the completion of the process
    - 0 is the starting value of the property
    - 1 is the final value

- Animation moving at linear speed

  → cubic-bezier(0,0,1,1)

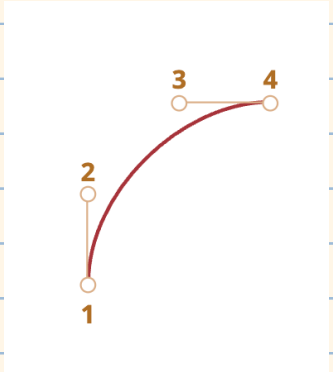
  → animation would move left to right w/a current speed




```
.train {
  left: 0;
  transition: left 5s cubic-bezier(0, 0, 1, 1);
  /* click on a train sets left to 450px, thus triggering the animation */
}
```

SUBJECT: transition-timing-function DATE:

- Animation that slows down



⟶ cubic-bezier(0.0, 0.5, 0.5, 1.0)

⟶ the process starts fast (curve soars up high), then becomes slower & slower

```
.train {
  left: 0;
  transition: left 5s cubic-bezier(0, .5, .5, 1);
  /* click on a train sets left to 450px, thus triggering the animation */
}
```

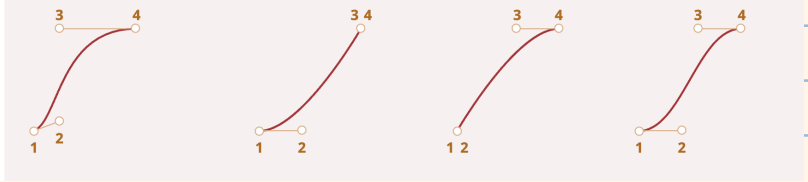
Built-in cubic-bezier curves

- linear ⟶ (0, 0, 1, 1)
- ease ⟶ (0.25, 0.1, 0.25, 1.0)
- ease-in ⟶ (0.42, 0, 1.0, 1.0)
- ease-out ⟶ (0, 0, 0.58, 1.0)
- ease-in-out ⟶ (0.42, 0, 0.58, 1.0)

SUBJECT: Bezier Curves DATE:

ease is used by default if there's no timing function

ease	ease-in	ease-out	ease-in-out
(0.25, 0.1, 0.25, 1.0)	(0.42, 0, 1.0, 1.0)	(0, 0, 0.58, 1.0)	(0.42, 0, 0.58, 1.0)



uses the ease-out shorthand ↘

```
.train {
  left: 0;
  transition: left 5s ease-out;
  /* same as transition: left 5s cubic-bezier(0, .5, .5, 1); */
}
```

- a bezier curve can make an animation exceed its range

⟶ the y-coord can be negative or very large

o this makes the curve extend either very low or high so the animation goes beyond its normal range

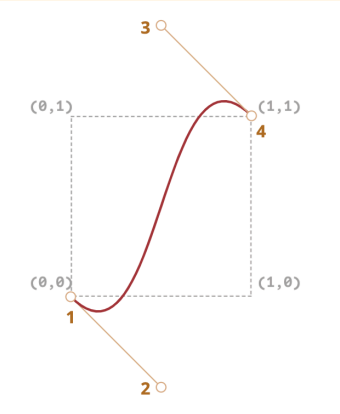
```
.train {
  left: 100px;
  transition: left 5s cubic-bezier(.5, -1, .5, 2);
  /* click on a train sets left to 450px */
}
```

↳ looks as if left should animate from 100px to 400px...

SUBJECT: Bezier Curves DATE:

Exceeding an animation range w/a bezier curve, continued:

```
.train {
  left: 100px;
  transition: left 5s cubic-bezier(.5, -1, .5, 2);
  /* click on a train sets left to 450px */
}
```



↳ When animation is clicked, it will:

- move back so left < 100px
- then move forward beyond 400 px
- move back again to 400 px

- y measures "the completion of the animation process"
  - y = 0 corresponds to starting property value
  - y = 1 corresponds to ending property value

→ So, y < 0 moves beyond starting point & y > 1 past final point.

- Tools for making bezier curves:
  - https://cubic-bezier.com
  - open dev tools on Mac with Cmd + Opt + I
  - click the icon that should be next to CSS properties that have cubic-beziers

SUBJECT: Steps Function DATE:

Steps Function (to use on transitions)

- Syntax ⟶ steps(number of steps[, start/end])
- splits a transition into multiple steps

- Example: will make a timer so the digits in #stripe appear one by one
  ⟶ must hide #stripe outside of #digit using overflow: hidden
  ⟶ #stripe needs to be shifted to the left step-by-step

```html
<div id="digit">
  <div id="stripe">0123456789</div>
</div>
```

```css
#stripe.animate {
  transform: translate(-90%);
  transition: transform 9s steps(9, start);
}
```

- The 2nd step function argument must be start or end

  ⟶ Start: The first step in the animation is made instantly
  - the animation will continue to be changed at the start of each new second

  ⟶ end: Nothing changes during the first second, creating a slight delay
  - changes will be applied at the end of each second

SUBJECT: Event "transitioned" DATE:

Event "transitioned"

- triggers when CSS animation ends
  - ⟶ used to perform an action once animation is done, or to join animations

- Example that moves ⟶ animation to one end of page. Then, it flips directions & moves to the other end. The animation moves further & further to right each time the transition finishes & go() is re-run.

```
1  boat.onclick = function() {
2    //...
3    let times = 1;
4
5    function go() {
6      if (times % 2) {
7        // sail to the right
8        boat.classList.remove('back');
9        boat.style.marginLeft = 100 * times + 200 + 'px';
10     } else {
11       // sail to the left
12       boat.classList.add('back');
13       boat.style.marginLeft = 100 * times - 200 + 'px';
14     }
15
16   }
17
18   go();
19
20   boat.addEventListener('transitionend', function() {
21     times++;
22     go();
23   });
24 };
```

- transitioned event object properties:
  - event.propertyName ⟶ the property that has finished animating
  - event.elapsedTime ⟶ the time (in seconds) the animation took w/o transition-delay

SUBJECT: Keyframes DATE:

Keyframes (a css rule)

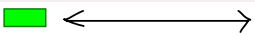
- @keyframes joins animations together
- specifies the "name" of the animation & rules
  → rules = what, when, and where to animate
- animation property is used to attach the animation to the element & specify any additional parameters

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

<style>
  @keyframes go-left-right {        /* give it a name: "go-left-right" */
    from { left: 0px; }             /* animate from left: 0px */
    to { left: calc(100% - 50px); } /* animate to left: 100%-50px */
  }

  .progress {
    animation: go-left-right 3s infinite alternate;
    /* apply the animation "go-left-right" to the element
       duration 3 seconds
       number of times: infinite
       alternate direction every time
    */

    position: relative;
    border: 2px solid green;
    width: 50px;
    height: 20px;
    background: lime;
  }
</style>
```



✳ @keyframes not used often - unless animation in constant motion

SUBJECT: Performance DATE:

Performance

- Most css properties can be animated b/c most are numeric values
  ⟶ animation able to alter numbers frame by frame

- The browser goes through 3 steps to render changes in style:

  1. layout – recompute element geometry and position

  2. paint – recompute how everything should look at their places

  3. composite – render final results into pixels onscreen & apply any css transforms

- load time is faster if the properties being animated skip layout step (skipping paint as well is even better)

- transform property is quick b/c browsers apply it "on top" of existing layout & paint calculations

  ⟶ changes (animation) of transform don't trigger layout/print

SUBJECT: Performance DATE:

- transform is efficient w/many functionalities
- Don't use things like left/margin-left properties

  ⟶ better to use transform: translateX(...), transform: scale, etc.

- Opacity never triggers layout either
- Most needs are met by pairing transform w/opacity
- clicking #boat ⟶ adds .move class to make animation move to the right & disappear

```
<img src="https://js.cx/clipart/boat.png" id="boat">

<style>
#boat {
  cursor: pointer;
  transition: transform 2s ease-in-out, opacity 2s ease-in-out;
}

.move {
  transform: translateX(300px);
  opacity: 0;
}
</style>
<script>
  boat.onclick = () => boat.classList.add('move');
</script>
```

SUBJECT: Summary DATE:

Example using @Keyframes:

```
<h2 onclick="this.classList.toggle('animated')">click me to start / stop</h2>
<style>
  .animated {
    animation: hello-goodbye 1.8s infinite;
    width: fit-content;
  }
  @keyframes hello-goodbye {
    0% {
      transform: translateY(-60px) rotateX(0.7turn);
      opacity: 0;
    }
    50% {
      transform: none;
      opacity: 1;
    }
    100% {
      transform: translateX(230px) rotateZ(90deg) scale(0.5);
      opacity: 0;
    }
  }
</style>
```

Summary

CSS animations allow smoothly (or step-by-step) animated changes of one or multiple CSS properties.

They are good for most animation tasks. We're also able to use JavaScript for animations, the next chapter is devoted to that.

Limitations of CSS animations compared to JavaScript animations:

Merits

- Simple things done simply.
- Fast and lightweight for CPU.

Demerits

- JavaScript animations are flexible. They can implement any animation logic, like an "explosion" of an element.
- Not just property changes. We can create new elements in JavaScript as part of the animation.

In early examples in this chapter, we animate `font-size`, `left`, `width`, `height`, etc. In real life projects, we should use `transform: scale()` and `transform: translate()` for better performance.

The majority of animations can be implemented using CSS as described in this chapter. And the `transitionend` event allows JavaScript to be run after the animation, so it integrates fine with the code.