CSS Animation

A Brief History of Web Animation

Throughout the 2000s, animation on the web was mostly done in Flash, a multimedia authoring program owned by Adobe.

Flash was notoriously resource-intensive. (I remember some Flash games would often crash the computers in grade school.) Flash support was never included on the iPhone for this reason, and as more people surfed the web on their phones, websites stopped using Flash altogether.

In 2009, WebKit, the browser engine developed by Apple, announced that it had implemented CSS Animations, and other browsers soon followed.

What are CSS Animations?

CSS animations allow an element to gradually change style.

Keyframes

Every CSS animation needs a set of keyframes.

Here, we create a keyframe rule for an animation named grow.

The animation will make some element's height start at 0 and end at 80px.

In your CSS:

```
@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}
```

Properties

Next, we bind the animation to an element.

Here, we specify that every element with the class *plant* will run the animation *grow* for 6 seconds.

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}
```

Properties

This is the bare minimum you need to create a CSS animation.

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}
```

Properties

Other animation properties:

animation-delay offsets the beginning of the animation.

Here, we tell CSS to wait 2 seconds before starting *grow*.

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: 2s;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}
```

Properties

Other animation properties:

animation-delay offsets the beginning of the animation.

A negative animation-delay tells CSS to start *grow* immediately, but midway through the keyframes.

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: -2s;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}
```

Properties

Other animation properties:

animation-iteration-count defines how many times the animation should run.

Here, we tell CSS to run *grow* two times instead of one.

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: -2s;
  animation-iteration-count: 2;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}
```

Properties

Other animation properties:

animation-iteration-count defines how many times the animation should run.

We can also tell *grow* to run forever.

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: -2s;
  animation-iteration-count: infinite;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}
```

Properties

Other animation properties:

Normally an animation runs from the 0% keyframe to the 100% keyframe.

animation-direction lets you run the animation in *reverse*, *alternate* (0→100→0), or *alternate-reverse* (100→0→100).

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: -2s;
  animation-iteration-count: infinite;
  animation-direction: reverse;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
```

Properties

Other animation properties:

animation-timing-function lets you specify the "timing curve" between styles in each keyframe, similar to CSS transitions.

Here, we tell *grow* to happen at a steady pace.

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: -2s;
  animation-iteration-count: infinite;
  animation-direction: reverse;
  animation-timing-function: linear;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
```

Properties

Other animation properties:

animation-timing-function lets you specify the “timing curve” between styles in each keyframe, similar to CSS transitions.

Other possible values allow you to accelerate (ease), or stop and go (step).

In your CSS:

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: -2s;
  animation-iteration-count: infinite;
  animation-direction: reverse;
  animation-timing-function: ease-in;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
```

Keyframes

Your animation can have multiple keyframes.

Each keyframe can involve multiple style changes.

In your CSS:

```
@keyframes grow {
  0% {
    height: 0;
    background-color: SpringGreen;
  }
  50% {
    height: 20px;
    background-color: ForestGreen;
  }
  100% {
    height: 80px;
    background-color: DarkGreen;
  }
}
```

Multiple anims.

You can also apply multiple animations to the same element.

First, we start by defining two animations' keyframes.

```
@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}

@keyframes change-color {
  0% {
    background-color: springgreen;
  }
  100% {
    background-color: forestgreen;
  }
}
```

Multiple anims.

Then, we apply both animations to our plant element.

Here, we tell *grow* to run for 6 seconds and *change-color* to run for 4 seconds.

```
.plant {
  animation-name: grow, change-color;
  animation-duration: 6s, 4s;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}

@keyframes change-color {
  0% {
    background-color: springgreen;
  }
```

Multiple anims.

Order matters!

The order that you provide the animation-name values determines the order that you should provide the other properties.

```
.plant {
  animation-name: grow, change-color;
  animation-duration: 6s, 4s;
  animation-delay: 2s, 1s;
  animation-timing-function: ease, linear;
}

@keyframes grow {
  0% {
    height: 0;
  }
  100% {
    height: 80px;
  }
}

@keyframes change-color {
  0% {
```

Shorthand property

The animation shorthand property lets you apply an animation in one line.

This could be nice for keeping your code tidy, but it can also be hard to decipher later on, so use this decisively!

```
.plant {
  animation-name: grow;
  animation-duration: 6s;
  animation-delay: 2s;
  animation-timing-function: ease;
}
```

```
.plant {
  animation: grow 6s 2s ease;
}
```

CSS Animation
(with JS for randomness)

Randomness

We can use Javascript's random number generator to create animations that are different each time you refresh the page.

To start, let's define a CSS animation called *sway*, and apply it to an element with the class *plant*.

In your CSS:

```
.plant {
  animation-name: sway;
  animation-timing-function: ease;
  animation-direction: alternate;
}

@keyframes sway {
  0% {
    transform: rotate(-10deg);
  }
  100% {
    transform: rotate(10deg);
  }
}
```

Random animation-duration

In JS, we can assign a value to the plant element's other animation properties. Here, we are telling the plant to complete one *sway* cycle every 5 seconds.

In your JS:

```
var plant = document.querySelector(“.plant”);

plant.style[“animation-duration”] = 5 + “s”; // “5s”
```

Random animation-duration

On a windy day, the plant sways quickly, every 3 seconds. On a still day, the plant sways slowly, every 7 seconds. Any day is somewhere in between, so we'll get a random number from 3 to 7.

In your JS:

```
var plant = document.querySelector(".plant");

var number = 3 + Math.random() * 4; // random num from 3 to 7

plant.style["animation-duration"] = number + "s"; // "4.1827s"
```

Random animation-delay

What if we want to find the plant in a different place each time? Remember that you can start an animation midway through its keyframes by using a negative animation-delay value.

In your JS:

```
var plant = document.querySelector(“.plant”);

plant.style[“animation-delay”] = -2 + “s”; // “-2s”
```

Random animation-delay

So, if the plant's *sway* animation takes 5 seconds, we'll want to start the animation at an offset of something between -5 to 0 seconds.

In your JS:

```
var plant = document.querySelector(“.plant”);

var number = -1 * Math.random() * 5; // random num from -5 to 0

plant.style[“animation-delay”] = number + “s”; // “-1.8295s”
```

Random animation-duration and -delay

Keeping it simple, you can give the plant different random values for animation-duration and animation-delay…

In your JS:

```
var plant = document.querySelector(“.plant”);

var duration = 3 + Math.random() * 4; // random num from 3 to 7
plant.style[“animation-duration”] = duration + “s”;

var delay = -1 * Math.random() * 5; // random num from -5 to 0
plant.style[“animation-delay”] = delay + “s”; // “-1.8295s”
```

Random animation-duration and -delay

If you are a statistical stickler, you may notice that there are some unreachable combinations of duration and delay here. For true, uniform randomness, you'll want to make the delay relative to the duration…

In your JS:

```
var plant = document.querySelector(“.plant”);

var duration = 3 + Math.random() * 4; // random num from 3 to 7
plant.style[“animation-duration”] = duration + “s”; // “4.12s”

var delay = -1 * Math.random() * duration; // 0 to -4.12
plant.style[“animation-delay”] = delay + “s”; // “-2.68s”
```

Randomness for multiple elements

If you have multiple elements with the class *plant*, you can select all of the plants using document.querySelectorAll(), loop through each plant, and change its style.

In your JS:

```
var plants = document.querySelectorAll(“.plant”);

for (var i = 0; i < plants.length; i++) {
  var plant = plants[i]; // the i-th plant element
  var duration = 3 + Math.random() * 4;
  plant.style[“animation-duration”] = duration + “s”;
}
```

Reference

CSS Animations — Mozilla Developer Network