GSAP Timelines: Animating Multiple Elements in Sequence

nobledesktop.com/learn/javascript/gsap-timelines-animating-multiple-elements-in-sequence

Dan Rodney December 1, 2023

Learn how to animate multiple elements in a sequence using GSAP's timeline feature in this comprehensive JavaScript tutorial.

This exercise is excerpted from Noble Desktop's JavaScript for Front-End training materials and is compatible with JavaScript updates through 2023. To learn current skills in JavaScript with hands-on training, check out our Front-End Web Development Certificate, Full-Stack Web Development Certificate, and coding classes in-person and live online.

Topics Covered in This JavaScript Tutorial:

Animating Multiple Elements, Creating & Using a Timeline, Cleaning up the Syntax with Chaining, Timeline Repeat & RepeatDelay Properties

Exercise Preview

Exercise Overview

In this exercise, you'll learn how to animate multiple elements in a sequence, using GSAP's timeline feature.

Getting Started

1. For this exercise we'll be working with the **GSAP Timeline** folder located in **Desktop > Class Files > JavaScript Class**. Open that folder in your code editor if it allows you to (like Visual Studio Code does).
2. In your code editor, open **timeline.html** from the **GSAP Timeline** folder.
3. Preview **timeline.html** in Chrome.

   It's a static logo like we worked with in the previous exercise, but now we've added a tagline below.

4. Back in your code editor, let's get acquainted with the HTML. Look in the **body** tag and you'll see:

   ```
   <img id="logo" src="img/noble-desktop-logo.svg" ALT="Noble Desktop">
   <p id="tagline">Design. Code. Create.</p>
   ```

5. Note the IDs on both elements: **logo** and **tagline**. We'll be using those IDs to identify the elements.

Loading the GSAP JavaScripts

1. Before we start animating, we need to link to the GSAP JavaScript file. Add the following bold code before the closing `</body>` tag:

```
</main>
   <script src="js/gsap.min.js"></script>
</body>
```

2. Add an empty script tag below that, which is where we'll write our own code:

```
<script src="js/gsap.min.js"></script>
   <script></script>
</body>
```

Animating the First Element

1. As we did in the previous exercise, start by calling the gsap.from() method with the following bold code:

```
<script>
    gsap.from();
</script>
```

2. In the gsap.from() method, add the following bold parameters:

```
gsap.from('#logo', {duration:1, scale:0, y:100});
```

3. Save and reload the page in Chrome.

   The logo should move up slightly as it scales in.

Animating Additional Elements

After the logo is done animating, we want the tagline to appear with the same animation.

1. Back in your code editor, copy and paste the line of code so you have the following:

```
<script>
   gsap.from('#logo', {duration:1, scale:0, y:100});
   gsap.from('#logo', {duration:1, scale:0, y:100});
</script>
```

2. In the second line, change the element we're targeting to the tagline's ID (we gave it that ID in the provided HTML):

```
gsap.from('#logo', {duration:1, scale:0, y:100});
gsap.from('#tagline', {duration:1, scale:0, y:100});
```

3. Save and reload the page in Chrome.

   Both elements animate the same way, at the same time.

   Interesting, but not what we wanted. We want the tagline to **appear** after the logo is done animating. There are a couple ways to accomplish this. First let's use a delay.

4. Switch back to your code editor.
5. Add the delay option shown below in bold (don't miss the comma before it!)

```
gsap.from('#logo', {duration:1, scale:0, y:100});
gsap.from('#tagline', {duration:1, scale:0, y:100, delay:1});
```

6. Save and reload the page in Chrome.

   That's better. The logo takes one second to animation, and the tagline has a delay of one second, so it starts as soon as the logo is done.

7. Switch back to your code editor.
8. Let's tweak this animation slightly. Let's make the tagline move down instead of up. Add a minus sign in front of the y value:

```
gsap.from('#tagline', {duration:1, scale:0, y:-100, delay:1});
```

9. Save and reload the page in Chrome.

   That's more interesting.

Creating & Using a Timeline

Sequencing the two animations this way worked, however there are downsides to this approach. Imagine animating 10 different elements. Changing the duration or delay of any tween will mess up the sequence. Another downside is that these types of sequences can't be paused, reversed, or controlled in any manner. All the tweens are running independently of each other. Luckily GSAP has a solution: a timeline.

1. Return to your code editor.
2. Let's get started creating our first timeline. Add the following bold code to instantiate a new timeline:

```
let tl = gsap.timeline();
gsap.from('#logo', {duration:1, scale:0, y:100});
gsap.from('#tagline', {duration:1, scale:0, y:-100, delay:1});
```

3. Replace `gsap` with `tl` (the name we gave to our timeline):

```
let tl = gsap.timeline();
tl.from('#logo', {duration:1, scale:0, y:100});
tl.from('#tagline', {duration:1, scale:0, y:-100, delay:1});
```

4. Save and reload the page in Chrome.

   The timing is a bit different now. The logo still takes one second to animate in, but there's now a delay of one second before the tagline appears. That's because each element you add to a timeline happens sequentially, one after another!

   We put the logo into the timeline first, so it happens first. Then we put the tagline in second so it happens after the previous animation is complete. We no longer need a delay, unless we wanted time between the elements.

5. Delete the tagline's delay (and comma before it) so you end up with:

```
let tl = gsap.timeline();
tl.from('#logo', {duration:1, scale:0, y:100});
tl.from('#tagline', {duration:1, scale:0, y:-100});
```

6. Save and reload the page in Chrome.

   Now the tagline will animate as soon as the logo is done!

Cleaning up the Syntax with Chaining

We can make this code even cleaner by chaining the animations. In short (without including vars in the example) you'll use the syntax `tl.from().from().from();`

In the chained animation syntax:

- We only use the name of the timeline once at the beginning.
- We only have one semicolon at the very end (which is optional).
- We can put each from() on a different line to help readability.

2. Back in your code editor, delete the **tl** from the beginning of the from() lines, so you end up with:

```
let tl = gsap.timeline();.from('#logo', {duration:1, scale:0,
y:100});.from('#tagline', {duration:1, scale:0, y:-100});
```

3. Delete the **semicolons ;** from the end of the from() lines, so you end up with:

```
let tl = gsap.timeline();.from('#logo', {duration:1, scale:0,
y:100}).from('#tagline', {duration:1, scale:0, y:-100})
```

4. Lastly, put the timeline name back in at the beginning, which we can do on it's own line:

```
let tl = gsap.timeline();
tl.from('#logo', {duration:1, scale:0, y:100}).from('#tagline', {duration:1,
scale:0, y:-100})
```

5. Save and reload the page in Chrome.

   The animation should still function as it did before. You just have cleaner code.

Adding More Elements to the Timeline

Now that we have a timeline, each animation will display in the order that we list them in the code. So adding a new animation is easier because we don't have to worry about delays.

In the previous exercise, we animated the logo so it scaled and rotated at the same time. What if we want it to scale and then rotate? We can add that as the second animation in our timeline.

1. Back in your code editor, below the logo animation add the following bold line of code for the new animation. Notice this time we're using a **to()** animation. This will start with the current appearance, and animate **to** the desired appearance:

   ```
   .from('#logo', {duration:1, scale:0, y:100})
   .to('#logo', {duration:1, rotate:360}).from('#tagline', {duration:1, scale:0,
   y:-100})
   ```

2. Save and reload the page in Chrome.

   Notice the sequence of the animation matches the order of our code:
   - First the logo scales up
   - Then the logo rotates
   - Then the tagline scales up

Adjusting Timing with the Position Parameter

It's a little monotonous having each tween start the moment the last one ends. What if we want to offset the start times of each tween? For greater flexibility, GSAP's position parameter allows us to adjust the start time of tweens.

How do we access the position parameter? The timeline's **from()** and **to()** methods accepts a position value as the 3rd parameter. The format is:

```
tl.from(target, {vars}, position)
```

To see how this works, let's focus on the tween that rotates the logo. Lets start the rotation 1 second after the logo animation ends.

1. Back in your code editor, add the following bold code to the logo's rotate tween and don't miss the comma!

   ```
   .from('#logo', {duration:1, scale:0, y:100}).to('#logo', {duration:1,
   rotate:360}, '+=1').from('#tagline', {duration:1, scale:0, y:-100})
   ```

   NOTE: This string value is relative to the end of the previous tween, meaning this tween will start 1 second **after** the previous tween is done.

2. Save and reload the page in Chrome.

   Notice the logo scales up, there's a 1 second pause where nothing happens, and then the logo rotates.

3. What if we want the logo to start rotating as it's scaling up? We can use a negative value instead of a positive value. Back in your code editor, make the following change shown in bold:

   ```
   .to('#logo', {duration:1, rotate:360}, '-=.5')
   ```

4. Save and reload the page in Chrome.

   Notice the rotation starts a little after the scaling starts, so for some of the time both the scaling and rotation are happening at the same time.

5. What if we wanted the tagline to start exactly 3 seconds after the start of the timeline, rather than being in the sequence of elements? Instead of using `+=` and `-=` which are relative to end the previous animation, we can use an absolute number (which is not a string).

   In the tagline tween, add the following bold code and don't miss the comma!

   ```
   .from('#tagline', {duration:1, scale:0, y:-100}, 3)
   ```

6. Save and reload the page in Chrome.

   Notice the tagline starts animating 3 seconds after the animation starts.

7. Let's say we want the tagline to animate at exactly the same time as the previous tween (which is the logo rotating).

   In the tagline tween, change the position parameter as shown below:

   ```
   .from('#tagline', {duration:1, scale:0, y:-100}, '<')
   ```

8. Save and reload the page in Chrome.

   The tagline should start animating at exactly the same time as the logo rotates.

   NOTE: We could also use `'<2'` to start 2 seconds after the previous animation starts. To learn more refer to greensock.com/position-parameter

Repeating the Animation

1. One of the benefits of using a timeline, is we can repeat the entire animation. Inside the gsap.timeline() method, add the following bold code to make it repeat twice:

   ```
   let tl = gsap.timeline( {repeat:2} );
   ```

2. Save and reload the page in Chrome.

   Notice it plays once, and then repeats twice. So it plays a total of 3 times.

3. The animation repeats immediately, but it would be nice to see the final appearance for a little while before it repeats. Back in your code editor add the following bold code, and don't miss the comma!

   ```
   let tl = gsap.timeline( {repeat:2, repeatDelay:2} );
   ```

4. Save and reload the page in Chrome.

   Now after the animation plays, it should pause for 2 seconds before repeating.

5. Back in your code editor, let's make it repeat forever by changing the repeat to **negative 1**:

   ```
   let tl = gsap.timeline( {repeat:-1, repeatDelay:2} );
   ```

6. Save and reload the page in Chrome.

   Now it will play, pause for 2 seconds, repeat, and keep doing that forever!

Safari Bug (on MacOS and IOS)

In Safari when the timeline repeats, an imprint of the logo is left behind as it starts animating a new logo, giving us 2 logos. Yuck! This glitch doesn't affect all GSAP animations, and we found a fix with CSS **will-change**.

MDN web docs: "The **will-change** CSS property hints to browsers how an element is expected to change. Browsers may set up optimizations before an element is actually changed. Warning: will-change is intended to be used as a last resort, to try to deal with existing performance problems."

Behind the scenes GSAP animations use a CSS **transform**. We can tell browsers that will change as shown below:

```
#logo {
   will-change: transform;
```

Code Omitted To Save Space

```
}
```

Dan Rodney has been a designer and web developer for over 20 years. He creates coursework for Noble Desktop and teaches classes. In his spare time Dan also writes scripts for InDesign (Make Book Jacket, Proper Fraction Pro, and more). Dan teaches just about anything web, video, or print related: HTML, CSS, JavaScript, Figma, Adobe XD, After Effects, Premiere Pro, Photoshop, Illustrator, InDesign, and more.