# GreenSock Animation Platform (GSAP) Guide

**Credits to GreenSock Official Documentation:** https://greensock.com/docs/v3

## What is GSAP?

GreenSock Animation Platform (or GSAP for short) is essentially a set of small JavaScript files which adds tools to make creating animations easier. It handles all the nitty gritty components of incrementally changing property values, which is what animation really is, so users can focus on the animation. GSAP is also extremely flexible and can add animation to basically anything JavaScript can touch. It's also super fast (said to be up to 20 times faster than jQuery) so there shouldn't really be any loading time downsides to using GSAP.

## Getting Started

There are many different ways to get GSAP but the easiest way is to load it from a CDN. This is similar to how we include Bootstrap but instead of a `<link ...>` tag we use a `<script ...>` tag because GSAP is made up of JavaScript files. Simply place this line of code right before the ending body tag, but before any other JavaScript files you wish to include.

```
<body>
    ...

    <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.2.4/gsap.min.js"></script>
    <!-- Any other JavaScript files you wish to add -->
</body>
```

## What exactly is a "Tween"?

In GSAP, a Tween is what does all the animation work and handles the property value changes. All you need to do is pass the target(s) (things you wish to animate), a duration, and the property or properties you wish to animate. The Tween will figure out what property values should be applied at each point in time.

Here are common methods for creating Tweens:

- `gsap.to()`
- `gsap.from()`

Let's go through each of these to see how we can make simple animations.

### `gsap.to()`

Let's first look at an example usage of `gsap.to()`.

#### HTML

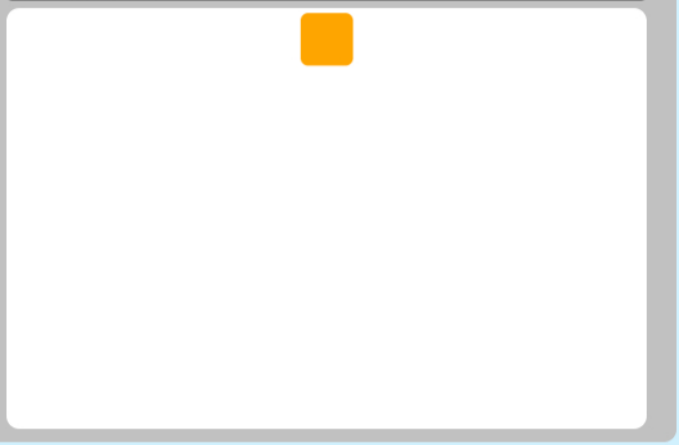
```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Intro to Tweens
        </title>

        <style>
            .box{
                background: orange;
                width: 7rem;
                height: 7rem;
                border-radius: 1rem;
                margin: auto;
            }
        </style>
    </head>
    <body>
        <div class="box"></div>

        <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.2.4/gsap.min.js"></script>
        <script src="script.js"></script>
```

```
    </body>
</html>
```

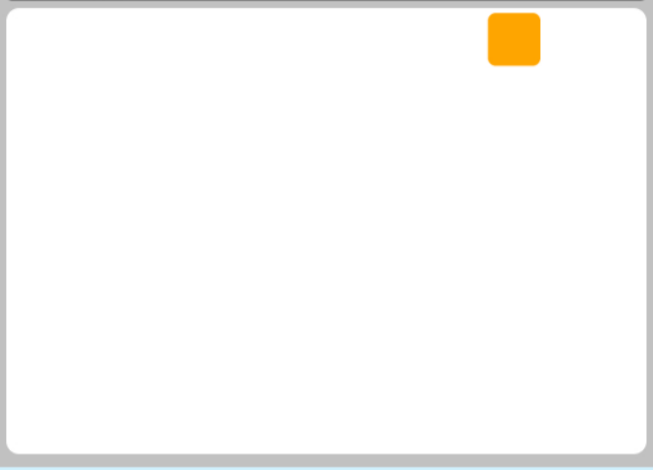
### JavaScript

```
gsap.to(".box", {duration: 1, x: 300});
```

### Initial Position



### Final Position



### Resulting HTML Upon Inspection

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Intro to Tweens
        </title>

        <style>
            .box{
                background: orange;
                width: 7rem;
                height: 7rem;
                border-radius: 1rem;
                margin: auto;
            }
        </style>
    </head>
    <body>
        <div class="box" style="transform: translate(300px, 0px);"></div>

        <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.2.4/gsap.min.js"></script>
        <script src="script.js"></script>
    </body>
</html>
```

Looking at the example above, we can see that `gsap.to()` moves elements **TO** another position or state (in this case moved in the x-direction +100 pixels). As you can see, we need to first pass the target(s), which is in this case is the class `box`. If we wanted to pass multiple targets, we would simply group them in a string and separate them with commas: `".box, #another-element"`. After passing the target, we then need to pass the variables in the form of an object with property-value pairs we wish to animate to.

Common variable properties we can animate include (BUT ARE NOT LIMITED TO):

- `duration`
- `x`
- `y`
- `opacity`
- `scale`
- `rotation`

## `gsap.from()`

Now here's an example usage of `gsap.from()`.

### HTML

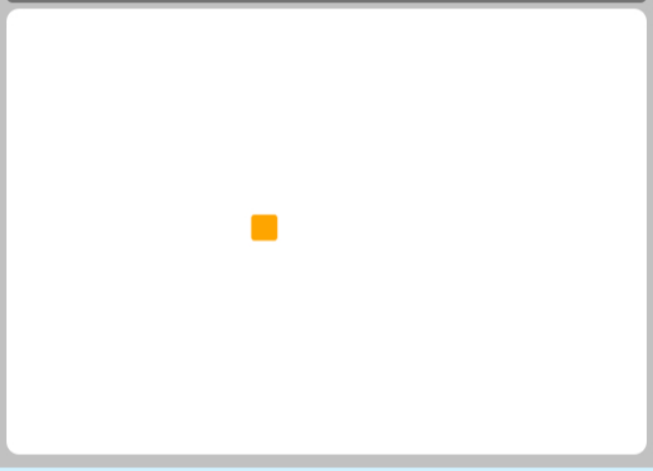
```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Intro to Tweens
        </title>
```

```
        <style>
            .box{
                background: orange;
                width: 7rem;
                height: 7rem;
                border-radius: 1rem;
                margin: auto;
            }
        </style>
    </head>
    <body>
        <div class="box"></div>

        <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.2.4/gsap.min.js"></script>
        <script src="script.js"></script>
    </body>
</html>
```

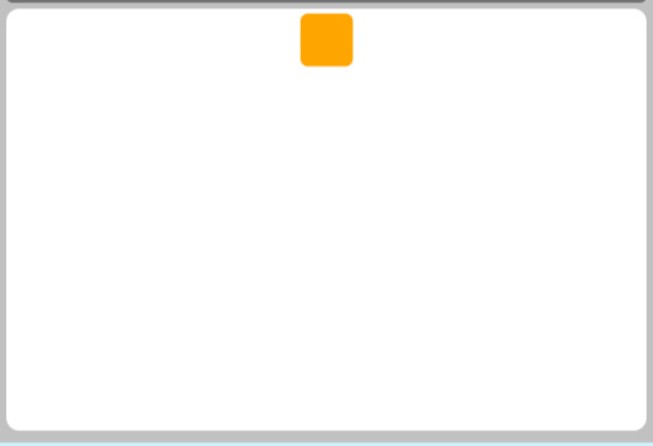
### JavaScript

```
gsap.from(".box", {duration: 1, x: "-100", y: 300, scale: 0.5});
```

### Initial Position



### Final Position



### Initial HTML Upon Inspection

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Intro to Tweens
        </title>

        <style>
            .box{
                background: orange;
                width: 7rem;
                height: 7rem;
                border-radius: 1rem;
                margin: auto;
            }
        </style>
    </head>
    <body>
        <div class="box" style="transform: translate(-100px, 300px) scale(0.5, 0.5);"></div>

        <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.2.4/gsap.min.js"></script>
        <script src="script.js"></script>
    </body>
</html>
```

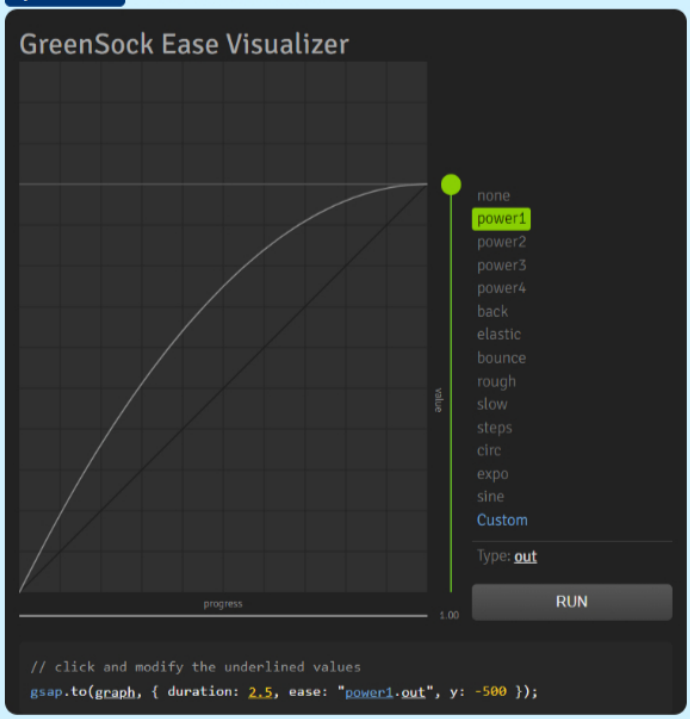
Similar to `gsap.to()`, `gsap.from()` animates elements **FROM** another position or state (in this case moved from -100 pixels in the x-direction, +300 pixels in the y-direction, and a scale of 0.5). Passing the targets and variables (as an object) works the same as with `gsap.to()` and we can also use the same properties listed above with `gsap.from()`.

## Easing

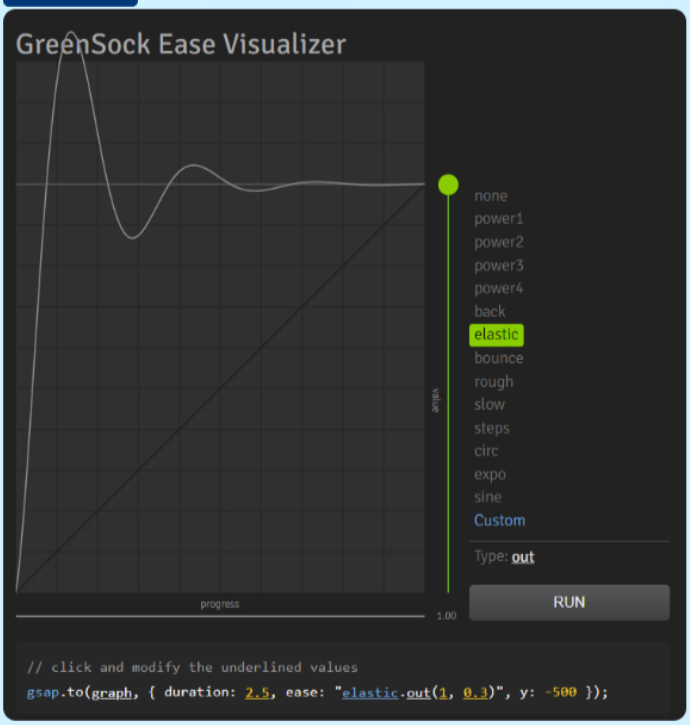
Eases are the main way to change the timing of Tweens. What this means is that we can effectively change how our animations might feel as you will see below. To change the ease used for an animation, we can add the `ease:"<insert_desired_ease>"` property to our variables object we pass to Tweens.

While it is fairly hard to show the effect eases have in a static format, we can use GreenSock's Ease Visualizer (found at https://greensock.com/docs/v3/Eases) to see these eases in a graphical format. GSAP's default ease is `"power1.out"` but another ease that I really love using is the `"elastic.out"` ease which adds some bouncyness to your animations. By default GSAP uses the `"<ease>.out"` version of eases so using the `"elastic"` ease would be the same as specifying `"elastic.out"`.

### `"power1"` Ease Visualization




### `"elastic"` Ease Visualization




## Staggering

Staggers are a way to animate groups of objects extremely easily. The easiest way to do this is by adding the `stagger:<time_offset>` property-value pair to our variables object we pass to a Tween. We can specify an offset amount between the start of each individual object's animation. While this is only going over the basics, staggering also can allow us to create 2d and 3d animations and play around with the ordering of animation of individual objects.

## Timeline

A GSAP Timeline allows us to order the sequence of animations and can be thought of as basically a container for Tweens and even other timelines. Lets look at the example code below to see how we can use timelines.

HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Intro to Tweens
        </title>

        <style>
            .box{
                background: orange;
                width: 7rem;
                height: 7rem;
                border-radius: 1rem;
                margin: auto;
            }
        </style>
    </head>
    <body>
        <div class="box"></div>

        <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.2.4/gsap.min.js"></script>
        <script src="script.js"></script>
    </body>
</html>
```

### JavaScript

```
var tl = gsap.timeline();

tl.from(".box", {duration: 1, x: "-100", y: 300, scale: 0.5});
tl.to(".box", {duration: 1, x: 300});
tl.to(".box", {duration: 1, y: 500, opacity: "50%", scale: 3});
```

### Initial Position

### Second Position

### Third Position

### Final Position



To create a timeline, we create a `gsap.timeline()` and assign it to a variable. We can then use methods of that timeline instead of using `gsap.to()` for example, it becomes `tl.to()`. Tweens are by default added to the end of the timeline so the animations happen in the order they are added in the example above.

Thank you for reading this basic guide to using GSAP! If you would like to learn more, check out GreenSock's Getting Started Page (https://greensock.com/get-started) and also their official documentation (https://greensock.com/docs/v3) where you can see in-depth demos and a bunch of advanced features.