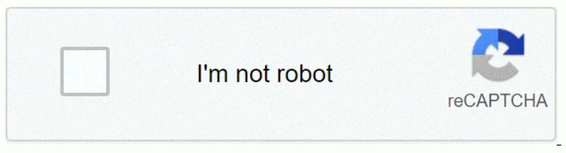
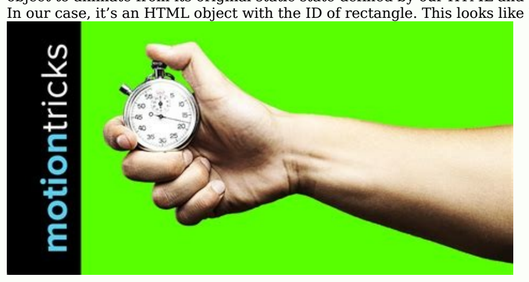


Continue

Gsap tutorial pdf

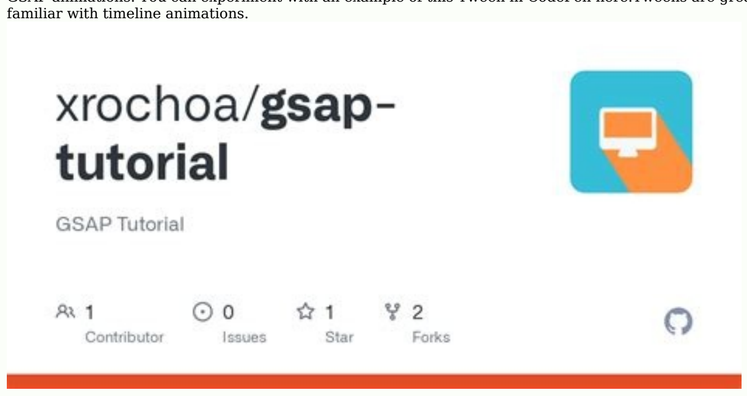
Animating with code may seem intimidating at first, but don’t worry, our platform was engineered to make it simple and intuitive. The menu to the left gives you access to every tool in the GreenSock API for HTML5. Select a tool to get an overview and list of every method and property. Every method and property has its own detail page too packed with descriptions and examples. Be sure to check out the tools in Plugins and Utilities packages too. You'll be amazed at what GSAP can do beyond its core animation abilities. Not sure where to look? Try the search feature. This chart shows which parts of GSAP are included inside of TweenMax, which are for Club GreenSock members only, and which reside on the public CDN. 82153571827.pdf All tools are linked to their docs for convenience.
Official GreenSock Training (videos and eBook) If you want to learn all of what GSAP can do check out our official video training Learn HTML5 Animation with GreenSock which includes over 5 hours of HD video. This course is packed with real-world projects and detailed step-by-step instructions. If you prefer to learn visually, you can watch this lesson in video format.If you want to follow along with the examples, you will need to create an app generated by the Vue CLI .What is GreenSock (GSAP)? GSAP is a Javascript animation library that helps developers script performant and engaging animations.GSAP is fast, robust with features and handles browser compatibility for us by default. And because it’s framework-agnostic, we can easily integrate and use it with Vue.To install the GSAP package, open a new terminal and navigate to your project folder, then run the following command.tip If you’re working in VSCode, you can go to Terminal > New Terminal to open a terminal or command prompt with the project folder already selected.Command: npm install gsapOnce the package has been installed, we should see it in the package.json file under “dependencies”.Alternatively, you can use Yarn to install the package.Command: yarn add gsapAs with any other dependency, we need to import GSAP into the file or component where we want to use it.Example: import gsapGSAP Methods GSAP provides us with multiple methods to create animations.Following are the most common methods.gsap.to allows us to define the ending values.
In other words, where we want to animate to and how it should happen.gsap.from allows us to define the starting values. In other words, a backwards animation.gsap.fromTo allows us to define both the start and end values.These methods take two parameters.The element that we want to animate.A vars object containing the properties to animate as well as animation options that specify the duration, easing functionality etc.GSAP uses short codes for 2D and 3D transform-related properties.CSSGSAPtranslateX(100px)x: 100translateY(100px)y: 100rotate(360deg)rotation: 360rotateX(360deg)rotationX: 360rotateY(360deg)rotationY: 360skewX(45deg)skewX: 45skewY(45deg)skewY: 45scale(2, 2)scale: 2scaleX(2)scaleX: 2scaleY(2)scaleY: 2translateX(-50%)xPercent: -50translateY(-50%)yPercent: -50To demonstrate, let’s set up a heading in the transition component with the before-enter and enter hooks.In the beforeEnter method, we’ll set the starting state of the heading. In other words, where the element will be when it starts the animation. In the enter method, we’ll use the gsap.to method and set the end state of the animation with a duration.Example: src/App.vueIf we run the example in the browser, the heading will slide down from the top of the page.How to delay the start of an animation GSAP allows us to specify a delay before the animation should begin with the delay property.To demonstrate, let’s add a 2 second delay to our example.Example: src/App.vueIf we run the example in the browser, the heading will slide down into the page after 2 seconds.How to repeat an animation We can repeat an animation by specifying the number of times we want it to repeat in the repeat property of the vars object.note The number of repetitions is added to the initial animation. A value of 1 would mean the animation happens once, then repeat once for a total of 2.tip To infinitely repeat the animation, we can use -1 as the value.To demonstrate, let’s add an infinite repeat to our animation. We’ll also remove the opacity and delay for now.Example: src/App.vueIf we run the example in the browser, the heading will slide down from the top and restart the animation once it reaches its end position.GSAP easing functions GSAP provides us with a lot of easing functions that we can use as a string value in the ease property of the vars object.We can select one from the list in the documentation to see a visual representation of how it will animate and also choose an easing type.There are three types of easings available.in will start slowly, then go faster toward the end of the animation.out will start fast, then slow down toward the end of the animation.inOut wil be fast through the middle, but start and end slowly.By default, GSAP uses the power1.out ease if we don’t specify our own.To demonstrate, let’s add the bounce animation with a type of out to our example.Example: src/App.vueIf we run the example in the browser, the heading will slide in from the top and bounce when it reaches its end position.How to create a stagger effect GSAP makes it easy for us to create a stagger effect with the stagger property of the vars object. Its value is the delay in seconds between the elements.To demonstrate, we’ll create some cards and iterate through them with a v-for based on id data properties. This time, we’ll use the from animation method and add the stagger property with a .1 second delay.tip Because the cards are in a container, we can use the card’s class to animate each element separately.Example: src/App.vueIf we run the example in the browser, the cards will slide down into the page one after another.How to indicate a completed animation Vue doesn’t know when a GSAP animation will complete. This can cause problems with our animations because transition hooks may fire when they’re not supposed to.To demonstrate, we’ll add the after-enter hook to our example and log a message to the console when it triggers. We’ll also increase the duration and stagger delay of the animation to make it clear what’s happening.Example: src/App.vueWhen we run the example in the browser, the console message will show before the animation completes. This would be true for all the leaving hooks as well.If we had any following animations, the whole thing would break.Luckily, we can tell GSAP when an animation completes with the onComplete property. Vue also gives us a second parameter in the transition hook that we can pass to the GSAP property.note We must add the first parameter to the hook, even if we don’t use it.Syntax: onComplete and done functionTo demonstrate, let’s add it to our example. We’ll remove the repeat property otherwise the after-enter hook will never be executed and we won’t see the console message.Example: src/App.vueThis time, the message will be printed to the console only after the animation is complete.Animations in lifecycle hooks instead of the transition component We can execute our animations inside a component’s lifecycle hooks . In that case, we don’t need to use the transition component or its hooks.To demonstrate, let’s remove the transition component from our example and move the GSAP animation over to the mounted lifecycle hook.Example: src/App.vueIf we run the example in the browser, everything still works as expected.Further Reading For more information on the topics covered in this lesson, please see the relevant sections below.GreenSock (GSAP) DocumentationGSAP CheatsheetGSAP Forums by Nicholas KramerA primer to creating timeline based animations without knowing JavaScriptIntroductionThe GreenSock Animation Platform (GSAP for short) is a powerful JavaScript library that enables front-end developers and designers to create robust timeline based animations. This allows for precise control for more involved animation sequences rather than the sometimes constraining keyframe and animation properties that CSS offers.The best part about this library is that it’s lightweight and easy to use.With GSAP, you can start creating engaging animations with little to no knowledge of JavaScript.This guide will show how to set up and use GSAP’s TweenMax feature and also dive into a bit of Club GreenSock’s DrawSVG plugin. Each of the examples below has a corresponding CodePen link so you can follow along in another tab.Getting StartedBefore coding, we first need to add the GSAP library to our HTML file.

To do this, you will need to grab the CDN link to the TweenMax library. You can find links to TweenMax and other GSAP CDNs here.Note: CDN stands for Content Delivery Network. This means that instead of hosting the JavaScript files on your site, an outside source like CloudFlare can host them for you.Once you have the CDN link, insert it in a That’s all you need to get started! If you’re using an online development environment like CodePen, you can install GSAP by editing the Pen Settings.Click the gear icon next to the JS text editor and search for TweenMax to install it in CodePenUnderstanding TweensTweens are the basic animation functions from within GSAP. To animate any HTML object, we must call the object, define the properties that we are going to animate, the duration of the animation, the animation’s easing, and any other parameters like delay timing.For example, if we were to change a red rectangle’s color to black while also moving it down and to the right, the Tween would look like this in JavaScript:TweenLite.to(“#rectangle”, 2, { left:100, top: 75, backgroundColor:"#000000", ease: Power4.easeIn});This tween gives us a rectangle that moves diagonally and changes color.Let’s break this down:TweenLite lets our JavaScript file know that we want to animate using GSAP. The .to method immediately after signifies that we want our object to animate from its original static state defined by our HTML and CSS to the final animated state defined by our JavaScript.You can use the .from method instead to reverse this. We’ll cover this a little later on in this article.Next, we define the object that we want to animate.
In our case, it’s an HTML object with the ID of rectangle. This looks like “#rectangle”, in our code.



We must make sure that we have our object wrapped in quotes and that we use the # to denote that we’re calling an ID. Any ID could go here as long as it’s an element defined in your HTML. mrs dalloway quotes with page numbers
Also, note that the comma following the end quote is important as well. Without it, the animation will not run.The 2, after the element’s ID defines the duration of the animation in seconds. So in this instance, we’re defining the animation’s duration as 2 seconds long.
If we wanted it to be a half-second long, we would change the value to 0.5, instead.The parameters inside the brackets represent any of the properties you’d like to animate. In this example, we’re animating the left top and background-color CSS properties.
Notice how each of these different properties use camelCase to call them instead of the typical hyphen notation used with CSS. You can add as many different properties here as you’d like as long as you separate them with commas after their value.The last property called is the animation's ease. GSAP comes packaged with a bunch of different easing options that you can add to your animations.In our animation above, we call the Power4 ease and have it set to easeIn to the animation. You can see the full range of easing options in the documentation here.

If you’re unfamiliar with easing, be sure to check out a previous article that explains different easing functions in depth.Finally, you must close the parenthesis and the brackets of the Tween to prevent any errors and allow the animation to run. Don’t forget to include the semicolon to end the JavaScript function.The Tween is the basic foundation for GSAP animations. You can experiment with an example of this Tween in CodePen here.Tweens are great if you want to do one-off animations but if you’d like to create multi-step sequences, timelines are the best alternative.Timeline AnimationsIf you’ve ever used an animation or prototyping software like After Effects or Principle, you’re already familiar with timeline animations.



Traditional timelines are usually a series of animations that occur one at a time or concurrently.
Timelines in GSAP are not any different.A visualization of a timeline in After Effects. GSAP timelines are not much different.To call a timeline, you must first define a variable at the top of your JavaScript file as a new TimelineLite:var tl = new TimelineLite;To break this line of code down piece by piece, know that var is short for variable. If you’re unfamiliar with what a variable is, think of it as shorthand for a larger piece of code. In this case, we defined a new variable as tl and set it equal to new TimelineLite. This means that every time that we input tl in our code, it will stand for a new TimelineLite.Note that we can substitute tl for any shorthand text we’d like.

I’m using tl because it’s short for timeline. This is useful because if we have multiple timelines in our file, we can give each one a unique variable to prevent confusion.Let’s now recreate our first animation using TimelineLite instead of TweenLite to see how this works.var tl = new TimelineLite;tl.to(“#rectangle”, 2, { x:100, y:75, backgroundColor: “#000000”, ease: Power4.easeIn})Notice how it’s rendering the exact same animation as the tween before.You’ll notice that this is not much different than our TweenLite animation from above. The only real difference is that instead of stating TweenLite.to we are using tl.to instead. You’ll also notice that we are now using x and y coordinates instead of left and top CSS properties.
These more or less behave the same way.Also, notice how there is no semicolon at the end of the parenthesis. This is because we are going to add a second animation to this timeline, tethering them together.For this second animation, let’s make the rectangle scale up 150% and turn gray after the first animation is complete. roludatirip.pdf To do this, we will add another block of code under our first animation. Altogether it will look like this:var tl = new TimelineLite;tl.to(“#rectangle”, 2, { x:100, y: 75, backgroundColor: “#000000”, ease: Power4.easeIn}).to(“#rectangle”, 1, { scaleX: 1.5, scaleY: 1.5, backgroundColor: “#454545”, ease: Back.easeOut.config(1.7)});We’re now tethering two animations together in a timeline.You can see that this second block of code doesn’t have the tl.to at the beginning of the timeline. Instead, it only has .to.This is because multiple animations in one timeline can be tethered together as long as there’s no semicolon separating them.A semicolon signifies the end of a timeline and should only be used at the end of the last animation within a timeline.You’ll also notice that we’re using two new properties, scaleX and scaleY. These properties behave exactly how they sound, they increase an object’s size by a percent amount. In this case, 1.5 is equivalent to 150%.Finally, we’re using a unique easing function here called Back.easeOut.config(1.7). This ease gives a natural rhythm to our animation by overshooting the intended value and then coming back down to the final value. We can see in this animation’s case how the rectangle grows slightly bigger than 150% and then scales itself back down afterward.Animating Multiple Objects with TimelineLiteTimelines are not limited to animating one object. You can animate multiple objects in a timeline by adding their corresponding IDs in different functions.For example, if we were to create an HTML object of a circle and have it fade in after our rectangle scales larger, our code would look like this:var tl = new TimelineLite;tl.to(“#rectangle”, 2, { x:100, y: 75, backgroundColor: “#000000”, ease: Power4.easeIn}).to(“#rectangle”, 1, { scaleX: 1.5, scaleY: 1.5, backgroundColor: “#454545”, ease: Back.easeOut.config(1.7)}).from(“#circle”, 1, { opacity: 0,});This latest block of code now has a circle fade in at the end of our animation.We’ve added a third code block to our animation that calls the circle.Also note how we’re now using the .from method. This means that our circle animation starts at 0% opacity and then goes to 100% opacity.You can see this in action when our animation has the circle hidden because it starts at 0% opacity. After the rectangle changes color and scales up, the circle fades in at 100% opacity, just as intended.You can see how TimelineLite works in this CodePen example here. I encourage you to try and add new elements to the HTML and try to create more complicated and unique sequences with the provided tools. You can also take a look at the full GSAP TimelineLite documentation here to learn about other methods and properties.DrawSVGAlong with the free TweenLite and TimelineLite features, GSAP also offers premium content that allows you to manipulate SVGs with ease.
Luckily, these plugins are available to play around with for free on CodePen by searching for them in the pen settings (the gear icon next to the JS text editor).The DrawSVG plugin makes it easy to animate the lines of an SVG. To show this, we’re going to have an SVG of a unicorn in a hoodie draw itself. You can follow along with the corresponding CodePen here.The final result of animating the SVG lines.First, we need to export our SVG and import it into our text editor.For a comprehensive breakdown on how to properly export SVGs, check out a previous article here.Next, we need to give each of our individual paths an ID so that we can call each one in our timeline. This may take some time if you have a complicated SVG with a series of different animating lines. For the sake of simplicity, I’m going to name the first path #unicorn1 and name the next path #unicorn2 and so on until they all have a unique ID.Now that all our paths have an ID, we can jump in and begin developing our timeline animation. mike meyers' comptia a guide to man
Like before, we need to create a variable will function as our TimelineLite variable:var unicorndraw = new TimelineLite();In this case, we’re going to be using the variable unicorndraw.The last step we need to do is to create a TimelineLite animation that calls each of the individual paths:unicorndraw.from(“#unicorn1, #unicorn2, #unicorn3, #unicorn4, #unicorn5, #unicorn6, #unicorn7, #unicorn8, #unicorn9, #unicorn10, #unicorn11, #unicorn12, #unicorn13, #unicorn14, #unicorn15, #unicorn16, #unicorn17, #unicorn18, #unicorn19, #unicorn20, #unicorn21, #unicorn22, #unicorn23, #unicorn24, #unicorn25, #unicorn26, #unicorn27, #unicorn28, #unicorn29, #unicorn30, #unicorn31, #unicorn32, #unicorn33, #unicorn34, #unicorn35, #unicorn36, #unicorn37, #unicorn38, #unicorn39, #unicorn40, #unicorn41, #unicorn42, #unicorn43, #unicorn44, #unicorn45, #unicorn46, #unicorn47, #unicorn48, #unicorn49, #unicorn50, #unicorn51, #unicorn52, #unicorn53, #unicorn54, #unicorn55, #unicorn56, #unicorn57, #unicorn58, #unicorn59, #unicorn60, #unicorn61, #unicorn62, #unicorn63”, 3, {drawSVG:”0", delay:”1"})You can see how this is like our previous .from TimelineLite animation from before. We’re calling our individual objects (in this case, we’re calling more than one at a time so that they all animate at once) and we define the duration of the animation as 3 seconds.The only difference is that inside the brackets, we’re now using drawSVG: “0”.
This calls the drawSVG plugin and defines each path to have a value of 0. Because we’re using a .from method, the paths start with a value of 0 and then animate to 100% in 3 seconds.You can play with different values to get a unique animation style.The other piece of code inside the brackets is delay: “1”. This determines how long the animation will wait to execute in seconds. In this case, we’re stating that the animation will wait 1 second before executing.This is the fastest way to get started with the drawSVG plugin but you can manipulate the values in many different ways to get some interesting effects. To learn more about this plugin, check out GSAP’s site.Final ThoughtsGSAP makes it easy to create and manipulate your own timeline animations even if you have little to no understanding of JavaScript. This was a small amount of the different animations you can do with GSAP. Check out Greensock’s site to learn more about the library and experiment with different animation techniques.Nicholas Kramer is a designer currently working at Barrel in New York City. He solves design problems for businesses by helping them simplify ideas and communicate their value to customers.Stay in Touch! Dribbble | LinkedIn | Website