8i Part #1 - Get started with CSS Animations

1. Get the most out of this course
2. Get acquainted with animation for the web
3. Use CSS transitions for simple animations
4. Use pseudo-selectors to trigger CSS transitions
5. Apply the 12 principles of animation to the web
6. Create multi-property CSS transitions
7. Use timing functions to create more natural animations

Quiz: Have you built a strong foundation in CSS animations?

8ii Part #2 - Translations, rotations, and opacity, oh my!

1. How do browsers render web pages?
2. Use the transform CSS property to ensure smooth animations
3. Change an element's anchor point using transform-origin
4. Analyze animation performance with Chrome DevTools
5. Create more peformant color animations using the CSS opacity property

8iii Part #3 - Make your animations dynamic

1. Create more complex animations using CSS @keyframes
2. Construct CSS animations using the CSS animation properties
3. Manipulate and reuse CSS animations
4. Refine your animations with Chrome's animation tool
5. Put it all together

%%

8iii Part #3 - Make your animations dynamic

%%

1. Create more complex animations using CSS @keyframes

- **Video notes**
  - -> combining multiple states
  - -> what they are, how to set them up, how to use them in the codebase
  - -> when you use a transition, you are going from one state to another
  - -> an animation made with keyframes consists of multiple states, an animation made with a transition consists of one state
  - -> this third of the course is just on keyframes <- combining multiple states into an animation
- **Taking the scenic route <- about the concept of using more than one state**
  - -> transitions have start values and end values
  - -> you are going in between two states, and can have x(t) curves for them
  - -> you can have more than one state in an animation
  - -> you can deliberately use @keyframes instead of a single transition, in order to (in this example) make a loading bar stutter more realistically - rather than use an ease in ease out x(t) curve
  - -> a transition is for an animation in between two states
  - -> a keyframed animation is one which combines multiple states to produce the animation
- **Stop and go: keyframes**
  - -> he's inserting keyframes into an animation which already exists and was created with a transition
  - -> the percentage of the animation completed when its value is realised
    - -> the end of the animation is 100% and the start is 0%, it's the progress value of the

animation
    - -> creating keyframes
        - -> keyframes are global
        - -> they aren't declared in a selector
        - -> in this example they are declared in their own section
            - @keyframes progress-bar{ <- the progress bar is the part of the webpage which is being animated in this example
            - 0% { <- when from when the animation is 0% complete, apply this transition
            - transform: scaleX(0); <- the properties / css styling you want for that section of the animation
            - }
            - 100% {
            - transform: scaleX(1);
            - }
            - }
        - -> you can do the same thing (in the case that there are two of them in this example), by replacing 0% with from and 100% with to
            - the state that the transition is coming from
            - the state that the transition is going to
- **Setting things in motion**
    - ***An example of how to add in a keyframe using Sass (indented css)***
        - -> he's defined a keyframe using @keyframes
        - -> now he's applying it to the active state of the button
        - -> animation-duration
        - -> this is like transition-duration <- s or ms units
        - -> then to use it
            -
            - .btn {
            - &:active { <- *he's gone into the Sass for the active state of the button*
            - & > .progress__bar {
            - transform: scaleX(1);
            - animation-name: progress-bar; *<- the code he's used under there is animation-name: progress-bar; -> progress-bar is the name of the keyframes which were added -> when the keyframes were defined this was the name which was used*
            - animation-duration: 1000ms; *<- the time it takes to cycle through the keyframes*
            - }
            - }
            - }
            -
            - @keyframes progress-bar{ *<- down here, it's called progress-bar -> this is the name of the keyframe which is the same one linked above to the active hover state of the animation*
            - 0% { *<- from 0 percent of the way through the animation, use this state*
            - transform: scaleX(0);
            - }
            - 100% { *<- go until this second state, 100% of the way through the animation*
            - transform: scaleX(1);
            - }
            - }
    - ***transitions vs keyframes for animations <- keyframes have higher specificity than transitions -> if the same element is targeted with an @keyframe and a transition, the***

animation from the @keyframe will take precedence

- -> transitions
  - transitions animate between two states
  - you need to tell it values which are held at those states
  - there is a trigger event in most cases which sets off the transition
- -> animations
  - you are telling it the keyframes and the durations
  - -> you don't have to e.g set the value of scaleX() for this
  - -> it's using the values of the keyframes at each stage of the animation
- -> if both are being used then keyframes overrides individual transformations which are being used
  - -> there can only be one transform property per class
  - -> the animations produced from the two methods, transitions and @keyframes behave differently
    - **in this example the animation made with @keyframes completely disappears after its been ran**

- ***this is a later updated version of the code in the example above***
  - .btn {
  - &:active {
  - & > .progress__bar {
  - animation-name: progress-bar; <- THIS LINE***
  - animation-duration: 1000ms; <- AND THIS LINE***
  - }
  - }
  - }
  -
  - @keyframes progress-bar{
  - 0% {
  - transform: scaleX(0);
  - }
  - 100% {
  - transform: scaleX(1);
  - }
  - }
- ***shorthand for the animation***
  - -> animation: progress-bar 1000ms; <- this is shorthand for the duration <- ***COMBINE INTO THIS LINE IN SHORTHAND

- **Break it up! — multiple keyframes**
  - -> x(t) curves in animations made with keyframes
  - -> the x is the percentage of the animation completed
  - -> the y is the name of the transition
  - -> you can come up with positions along that curve
  - -> he's taken the x(t) curve for the transition
    - -> then taken several coordinates along that curve
    - -> then split the entire transition up into keyframes with those difference along the curve for the original transition
    - -> those keyframes are overriding the initial code
  - -> the default x(t) curve is a cubic Bezier function
  - .btn {
  - &:active {
  - & > .progress__bar {
  - animation: progress-bar 1000ms;

    - }
    - }
    - }
    -
    - @keyframes progress-bar{
    - 0% {
    - transform: scaleX(0);
    - }
    - 17% { <- *these numbers for the keyframes correspond to those points along the cubic-Bezier function for the transition -> he's taken several coordinates along that curve which would be used for transitions, and then used those values to make keyframes from - so the entire animation is a more jittery loading bar filling in, rather than a smooth function*
    - transform: scaleX(.18);
    - }
    - 24% {
    - transform: scaleX(.4);
    - }
    - 46% {
    - transform: scaleX(.81);
    - }
    - 100% {
    - transform: scaleX(1);
    - }
    - }
    - -> it's like separating out an animation into different sections
    - -> timing functions and keyframes
    - -> if there are no timing functions then the default is an ease timing function
- **The more the merrier: multiple properties**
    - -> the next chapter is timing functions on keyframes
    - -> adding opacity to the keyframes
    - -> so, we have a progress bar
        - -> which is going from left to right and stuttering as it does so
        - -> the animation which moves it from left to right was created using the @keyframe method
        - -> each keyframe is a state -> which was chosen based off of a point from the more smooth x(t) graph for a transitions between two states
    - -> this section of the chapter is about adding in a gradient to the progress bar as it moves across the screen <- changing the opacity of each keyframe as the loading bar moves across the screen
        - -> in other words, how to transition the opacity of a single colour for colour transitions on animations made using keyframes
        - -> at the moment, the styling under the keyframes is only controlling the expansion of the progress bar and not its colour
    - -> how I would summarise what he's done
        - -> we have a loading bar -> that loading bar is moving across the page from LHS to RHS
        - -> that animation was done using keyframes
            - where we started before this section
                - -> he took an animation which was made using a single transition -> in other words, one which was made using the transition property in css / Sass (indented css)
                - -> then he took the x(t) curve for that, he took several points along the curve
                - -> it looked like a histogram, where you take a surve and divide it into discrete

chunks
    - -> the top of each one had a coordiante
    - -> he broke up the single transition into multiple of those coordinates
    - -> then he put them into keyframes
    - -> so the same transition (a smoothly loading bar) looked like a bar which had more stutter as it loaded
        - -> he did this to make it look more realistic as it loaded
    - -> that was everything leading up to this section / subsection in the chapter
    - -> that bar had the same colour as it moved across the screen
- the aim of this section
    - -> this section of the chapter was on how to make it become more opaque as it moved across the screen
        - in other words, the colour was the same -> but how to get it to start off as completely transparent and end as opaque as it moved across the screen
- how the did it
    - -> so we had the transitioning search bar -> and you could put in code under each keyframe for that
    - -> he set a default value for the opacity which was 0 -> in other words, transparent
    - -> then as the keyframes went along -> he set the last two to have an opacity of 1
    - -> so the keyframes would start off going from the default value which was set outside of the keyframe definition (an opacity of 0, completely transparent)
    - -> to transitioning to be opaque as they approached the keyframes towards the end of the animation, which were completely opaque
    - -> then once the loading bar got past those two keyframes at the end which were comepltey opqaue, then the opacity of the entire thing reached the deafult value which was set in the code <- it went transparent again
    - -> he didn't need to set the opacity of each keyframe
    - -> he just changed two of them at the end (for the last two keyframes aka states which the animation was transitioning through) and set a default value for the transparency of the loading bar outside of the code for the keyframes -> which was transparent
        - -> for the keyframes leading up to the ones at the end, he didn't need to set an opacity for those -> because they were transitioning towards the ones at the end anyway -> he just set the final value of the opacity on the keyframes towards the end and the default as the ones he wanted it to start from
            - -> then the browser would calculate the opacities of the keyframes at the beginning (without having to set them)
        - -> and then when the animation had ended, the opacity of the loading bar reset to the default value which was defined in the code (transparent)
- the final code for this was
    - .progress {
    - &__bar {
    - opacity: 0; <- *the default opacity is transparent*
    - }
    - }
    -
    - @keyframes progress-bar{
    - 0% {
    - transform: scaleX(0); <- *the browser is calculating the opacities for these*

keyframes by transitioning them in between the default opacity of the page and the opacity of the keyframes at the end of the animation -> without having to set the opacity of these frames, it's doing it anyway

```
    }
    17% {
        transform: scaleX(.18);
    }
    24% {
        transform: scaleX(.4);
    }
    46% {
        transform: scaleX(.81);
    }
    85%,100% { <- the last two keyframes have an opacity of 1
        opacity: 1;
    }
    100% {
        transform: scaleX(1); <- after this keyframe in the animation, the opacity of the entire loading bar is going back to the default opacity of the page -> if there are two batches of Sass for the same keyframe, then the last one will take precedence
    }
}
```

- **Let's recap!**
  - -> keyframes
    - more complex animations <- combining multiple states
      - at each of them you can change the properties of the element being animated
      - -> the syntax for this is @keyframes name-of-set {}
      - -> then inside those brackets its x% {}
        - -> x percentage of the way through the keyframe -> animation
        - -> then inside those brackets it's the styling which applies for that section of the animation
      - -> if you only use 0% and 100%
        - -> aka, before and after, then there are two states / frames
        - -> in which case -> you can use from and to
        - -> if there is no styling in the code for that section of the keyframe, then the default values defined in the rest of the code are used
        - -> the same property doesn't need to be defined in every keyframe
          - you can set its value in various different keyframes, and then the browser will create an animation which transitions between those values for the different keyframes where its value hasn't been defined
          - -> styling which is defined in single keyframes for an element will override the default styling for that element defined in the rest of the css / Sass (indented css)

2. Construct CSS animations using the CSS animation properties

- **Video notes <- see section 5. for these**
- **Transitions vs. animations <- long winded explanation about the difference between a transition and an animation (made using keyframes)**
  - -> you can add x(t) curves to individual keyframes, then use those keyframes for any element you want on the webpage
  - -> how I would summarise this is

- -> it's about the difference between an animation and a transition
- -> a transition is something which is transitioning between two states
  - -> the initial state is the webpage in its default state
  - -> the final state (in the examples he's used in this course) is the state of an element on the webpage while it's being interacted with by a user
  - -> the 'animation' is the transition which is going between those states
  - -> the final state only exists for as long as the user is interacting with the element which is triggering it
  - -> you have two states, and the final state has to exist -> its existance depends on wheather the user is interacting with that element in the specified way or not
- -> compare this to an animation
  - -> the point he's making is that animations made via transitions and keyframes are two very different things
  - -> an animation made using a keyframe has a triggering event -> for example the element being animated has had some sort of user interaction to trigger that animation
  - -> in this example - in both cases, that could trigger an animation made using a keyframe or a transition
  - -> but in the case where a keyframe has been used, the animation which has been set is a -> it's like a chain reaction of a set of different transitions
    - -> and once they've been started, they can't be stopped <- they can, be interrupted
    - -> but the final one doesn't depend on the thing which triggered the first one ending
    - -> the final state of that animation isn't the code for the element defined under its pseudo-class
  - -> with transitions, it's the before and after state of that element
  - -> with keyframes it's the before state which can be triggered by the same user interaction event with transitions
    - -> but it's ending when all of the keyframes -> the series of little transitions have been completed
    - -> not when the user interaction which triggered it ends

- **From the get-go**
  - -> the way I would summarise what he's said is
    - *what this section of the chapter is about*
      - -> with keyframes, rather than using user interactions to trigger animations, you can use the loading / reloading of the page as the event which triggers the animation
      - -> animations don't have to be triggered by an interaction event <- where the user is interacting with the element on the webpage
      - -> animation can be triggered by a multitude of different events -> and in this example, it's the loading / reloading of the page (rather than the user interacting with the animated element)
      - -> this section of the chapter is about how that can be done for an animation made using keyframes -> and the code which makes that possible
    - *the code which has made it possible to do this*
      - -> rather than using the active state of the loading bar to trigger the animation (in other words -> a user interaction with the element being animated on the webpage), he's changed the code to get rid of that active state
      - -> this is the Sass (indented css) which he's used
        - .progress {
        - &__bar {
        - animation: progress-bar 1000ms; <- *the animation is set to the*

progress-bar here, aka when the progress-bar is loaded, this is when the animation is triggered to start -> not when the user is interacting with it (not when it's in its active state). The progress bar is loaded when the page is refreshed -> so the trigger event for the animation is the reloading of the webpage

-
- }
- }
-
- @keyframes progress-bar{ *<- then everything in terms of its keyframes is the same*
- 0% {
- transform: scaleX(0);
- opacity: .1;
- }
- 17% {
- transform: scaleX(.18);
- }
- 24% {
- transform: scaleX(.4);
- }
- 46% {
- transform: scaleX(.81);
- }
- 85%,100% {
- opacity: 1;
- }
- 100% {
- transform: scaleX(1);
- }
- }
- -> he's then changed the html to get rid of the button which initially triggered the animation -> it's the same html as in the previous chapters, but just without the button
  - -> the html for this example is below
    -
    -
    -
    -
    -

- **Let's wait and see**
  - *the aim is to take an animation which is being triggered when the page reloads, and add in a delay -> so the page reloads, then we want the animation which that event tiggers to wait a while before it starts*
    - -> we're using shorthand notation for the code in Sass (indented css) to delay animations created using keyframes which are triggered by page reloads, rather than user interactions
    - -> in other words, the same animations as in the previous section of the chapter <- a chapter being 1., 2., 3. etc in this note structure
  - *this is an example of the animation-delay code in Sass (indented css) which is used to achieve this*
    - .delay-seconds {

        - animation-delay: 1s; <- *in this example, the property which is used to do this in Sass is animation-delay -> the animation is being triggered not by a user interaction which puts it into a pseudo-state -> it's being triggered by the reloading of the page, and it's been created using keyframes. This line of Sass is taking the animation which has been created in that context and saying -> when the page reloads (in other words, the event which would trigger the start of the animation, wait 1s before starting the animation). So -> when the page reloads, don't start the animation which would be triggered by that event until a second after the event happens*
        - }
        -
        - .delay-seconds {
        - animation-delay: 1000ms; <- *this is the same example as in the block of code above, but it's using ms rather than s -> this is to show that the units of that line of code are in s or ms*
        - }
        - -> the equivalent of this for transitions would be transition-delay, but in this case the line of Sass which is being used is animation-delay
    - *an example of the shorthand code for this*
        - -> .progress {
        - &__bar {
        - animation: progress-bar 1000ms 150ms; <- *progress-bar is telling it what the event which triggers the keyframe animation is. It's not the active state of that element (the event which triggers that animation is the progress-bar itself, in other words when that bar reloads -> when the page reloads, then trigger that animation on that bar). The second 1000ms <- this is telling it in shorthand notation how long the animation on that element is going to take. The last 150ms <- this number is the delay, it's shorthand for animation-delay. In other words, when the page reloads to trigger the animation, in between that event and the start of the animation - wait 150ms. Combine everything, and you're telling it: when the page reloads - then trigger the animation, spend this long running the animation, but in between the event which triggers the animation (the page reload) and the start of the animation, wait this long*
        - }
        - }
    - *limitations of this*
        - -> the page reloads, then there is a delay in between the page reload and the start of the animation
        - -> then the animation runs for the time which it's been set to run for
        - -> then once it's ran -> it just turns transparent -> to the default opacity of the page
        - -> once the animation is over, then we want -> in this example an image of a cat to show
            - rather than the animation just ending and then nothing being there being it does
- **We waited 1.15 seconds and all we got is this picture of a cat**
    - *-> the aim of this section is*
        - once the animation has ended, then show a picture
        - -> and
            - the animation which we are dealing with was created using keyframes
            - -> it's triggered when the page reloads
            - -> there is a delay in between the event which triggers it and when it starts
    - *-> how he's done this is*
        - *-> found the content and added it in with a new div in html*
            - -> first found a picture which he wants to show when the animation ends -> in this case it's a picture of a cat
                - -> unsplash.com or pexels.com <- for royalty free images

- -> then added it into the html
  - he's added a new div in html -> after the div of the loading bar being animated
  - he's edited that div for the picture -> with text to show over the image which he wants to show after the end of the animation

- -> *changed the styling of the photo in css (Sass, indented css)*
  - -> *to do this he's added the image in as a background photo for the animation, and moved that image to be in front of the element being animated*
    - -> the css he's used to do this is
      - .cat {
      - width: 50vw;
      - height: 30vw;
      - position: absolute;
      - overflow: hidden;
      - background-image: url("https://bit.ly/2XJJLKn"); <- *he's added in the picture as a background-photo*
      - background-size: cover;
      - background-position: -20%;
      - z-index: 1; <- *then he's moved that image to the front of the loading bar (the element being animated) in the 'stack' in the z direction*
      - display: flex;
      - flex-direction: column;
      - justify-content: space-between;
      - align-items: center;
      - padding: .1rem;
      - font-size: 4vw;
      - font-weight: 900;
      - color: white;
      - }
        - -> he's added in an empty div into html where he wants the image to go, after the div for the loading bar being animated
        - -> then he's used Sass (indented css) to inject the content for that photo into the html
        - -> that's a pseudo-element <- he's injected the html for the photo in as a pseudo-element because it's a part of the animation which isn't a default element of the page
  - -> *that image which he's added into the animation (which we want to show once it's finished) is now showing up in front of the loading bar which is being animated, so he's now adding in keyframes which make the picutre show at the end of the animation*
    - -> this entire section is about making that picture which is now in front of the loading bar being animated show at the end of that animation -> rather than during it
    - -> he's doing that using keyframes
    - -> the code he's written for this section of the chapter is for adding in those keyframes
    - -> the Sass (indented css) which he's used for this is below
      - @keyframes cat{ <- *the name of the keyframes are for the cat (photo) and aren't included as part of the animation keyframes for the loading bar -> this is an example of combining multiple sets of keyframes -> and then getting one set to run after the others*
      - 0% { <- *0% of the way through the loading bar animation, then change the position of the cat photo which is front of the loading bar -> move that*

picture far off the page to the left
        - transform: translateX(-9999px);
        - }
        - 100% {
        - transform: translateX(0); <- *then when the loading bar animation is at 100% and has completed, then take the cat photo in front of it which has been moved far off the page to left -> take that photo and now move it to be back directly in front of the loading bar*
        - }
    - }
    - *further commentary about the code for the cat div*
        - -> translateX is a function
        - -> the introduction of variables into the code in order to control the delay of the start of the animation
- -> *then he's adding in the code for that cat div to the code for the end of the loading bar*
    - -> in other words, the code in the previous section was for the individual cat div
    - -> the code in this section is for combining the code for that cat div with the code (Sass, indented css) for the loading bar being animated
    - -> this is done using the code below <- the combination of that cat div with the loading bar being animated
        - $prog-bar-dur: 1000ms; <- *variables at the top of the page, for dictating the timing and duration of the animation*
        - $prog-bar-delay: 150ms;
        -
        - $cat-delay: $prog-bar-dur + $prog-bar-delay; <- *we're saying the time for the total animation is the time for the bar to load plus the time for the animation which shows after it -> this is to combine animations which had been made with keyframes, we are combining an animation made with one keyframe with an animation made using another keyframe. This is saying the total time for both those animations to run is the time for the first one to run plus the time for the second one to run. The reason we're writing code which combines both of them is because one is starting when other ends -> and both of them are being animated on the same part of the webpage*
        -
        - .cat {
        - animation: cat 0ms $cat-delay; <- *this is the code for the image which is showing at the end of the loading bar animation (the css / Sass = indented css for it) -> it's shorthand. This reads: the thing we're animating is the cat keyframe defined below, then its delay is 0ms, in other words the animation takes 0ms to run, and then the last variable in that line is saying -> in between the trigger for the start of that animation and the when it acutally starts, wait this long -> wait for the amount of time which is stored in the variable called $cat-delay, which is defined elsewhere in the code*
        - }
        -
        -
        - @keyframes cat { <- *then this is the Sass (indented css) which defines that animation*
        - 0% {
        - transform: translateX(-9999px); <- *it has two parts -> the definition for this animation as in the example above, the annotations for this code would*

be the same as in that section in this note

```
    }
    100% {
        transform: translateX(0);
    }
}
```

- -> *the code for this animation isn't working, because the elements are just returning back to their default values*
  - -> the rest of this section of the chapter is just a more long winded explanation of this
  - -> the key points / thought processes / approaches are
    - -> commenting out section of code when something isn't working
    - -> playing around with the values of things in order to troubleshoot
    - -> he's changing different values and seeing how the behaviour of the elements being animated changes <- in order to figure out what is causing it to not work
      - -> increasing the time things take so that that those issues will be more visible
  - -> if you are combining several animations made using keyframes, and one starts at the end of the other, then the one which isn't being used at that moment in time will revert back to its default value
    - -> in this case, this is where the code is going wrong
    - -> this is also the case with delays
      - -> an event which triggers the start of an animation has happened
      - -> is that animation has a delay, then there will be a time in between the start of the animation and when the trigger event for that animation has happened
      - -> in that time period, the elements on the webpage for which that animation has been defined will take their default values
      - -> those values are often set in the code outside the definition of the keyframes which define the animations

- **Prologue/epilogue**
  - -> a common approach is to increase the duration of an animation to a large value which you wouldn't normally want it to have, but you would only have for the duration of the animation -> so that you can troubleshoot
    - -> in other words, you increase the duration of the animation in the code when you are creating it
    - -> so that changes are more visible and its behaviour is more visible
  - -> *how I would summarise what he's written*
    - -> *#1 the first is the concept of different fill modes when it comes to animations*
    - *-> #2 the second is how they apply to the example from the previous section*
    - *-> #3 then the third is adding in a time delay in between the end of the first animation and the start of the second*
  - ***-> #1 the concept of different fill modes when it comes to animations***
    - -> the animations in this context have been created / added in using keyframe in Sass (indented css)
    - -> the idea behind fill modes is - that when an animation is happening, you are telling the element on the webpage how to behave using the keyframes which define it
      - -> these are a series of states which have their own styling and are created in Sass
      - -> in other words, the animations you are dealing with in this case are a series of keyframes
      - -> they are telling you what the animation does when it is running

- -> the question is, what is happening before and after that animation is running
- -> that's what fill modes are
- -> are filling the time before and after that animation is running with styles which tell it how to run / behave

- -> so we have - before the animation runs
  - -> this is backwards
  - -> the names are relative to when the animation runs
  - -> so 'backwards' is referring to the behaviour of the animation backwards in time (backwards from when it is running)
  - -> and then 'forwards' is referring to the time after which the animation has been run, into the future
  - -> backwards -> this Sass targets the behaviour of the animation before the trigger which causes it to run has happened
  - -> forwards -> this code targets the behaviour of the animation after the last keyframe which defines it has ended
  - -> in other words
    - *the concept of the backwards and forwards region in relation to an animation which has been created using keyframes*
      - -> the time in between minus infinity and the trigger for when the animation starts <- that's 'backwards'
      - -> then the time in between the
        - -> the start of an animation is triggered when an event on the webpage happens
        - -> the end of the animation isn't triggered, it's happening when the last keyframe which defines the animation has been ran
        - -> then forwards is the time in between when the last keyframe which defines the behaviour of the animation has been ran -> the time in between that point and t equals plus infinity
          - -> so we are dealing with three time periods in relation to the animation
          - the first is backwards, then during (set in the keyframes which define the animation), then forwards (into the future after it has been ran)
    - *language in Sass which is being used to target the behaviour of the elements being animated in those regions of time*
      - -> the fill mode is using Sass to target the behaviour of the element being animated before / after the animation <- in other words, in its backwards and forwards regions
        - -> in those time periods, we can either set the styling of the element being animated to its default value for the page, or depending on which of the backwards or forwards period you are in - you can set the styling of that element equal to the values which it had in the instant at the end of the animation, or start (see diagram)

      - -> if you are targeting the styling of the element being animated in the first and third regions (before and after the animation happens), this is a fill-mode referred to as 'both'
        - -> just the first region <- this is backwards
        - -> just the third region <- this is forwards

                - -> just the one in the middle <- this is the animation itself, that region is being targeted using the keyframes which create it
            - -> to go from region 1 to 2, there is a trigger which happens <- and in this case, it's doesn't always have to begin when a user triggers the pseudo-state of an element in Sass (indented css), it can happen when the page reloads
- ***-> #2 the second is how they apply to the example from the previous section***
    - *using Sass (indented css) to target the behaviour of the element being animated in that region*
        - *not shorthand method*
            - -> there is the Sass, and then there is the shorthand notation for it
            - -> animation-fill-mode: both; <- *this Sass isn't the shorthand - in other words, it's animation-fill-mode - telling it what styles to fill those two time periods (or either of them) with. This - both value - is referring to the forwards and backwards regions. If none of them have been targeted in the code then the styling of the element being animated within those regions of time gets set back to their default values, which are set elsewhere in the Sass (indented css)*
        - *shorthand method*
            - -> this can be combined with the other code used to format the animation, in shorthand notation
            - -> this is in the example below
                - animation: progress-bar $prog-bar-dur $prog-bar-delay both;
                - *-> from left to right: progress-bar <- this is the name of the keyframe which is defining the animation. When the bar loads, thisis the trigger that starts that animation*
                - *-> next: the $prog-bar-dur $prog-bar-delay <- both of these are variables whose values have been defined in the code beforehand. They tell the browser how long the keyframed animation should take and the delay in between the end of the event which triggers it and when it actually starts should be*
                - *-> the last one is both <- this is telling the browser to do two things. The first is, set the styling of the element being animated when it is in region 1 of the diagram above (in this note) - the styling in that region is the same as the styling of the animation in the instance when it is first triggered. And, likewise for region 3 of the diagram above (the instance in this case the last keyframe which defines it)*
- ***-> #3 then the third is adding in a time delay in between the end of the first animation and the start of the second***
    - *the idea behind this section of the chapter*
        - -> now we're going back to the example where we have two animations which have been defined using keyframes and are being combined on the same place in the webpage
        - -> the first is the loading bar and the second is the picture of the cat which shows after it loads
        - -> is part of the subsection is about adding a time delay in between the end of the first animation and the start of the next, which he has done using Sass (indented css)
    - *how he has approached this*
        - -> inthe Sass which is combining both of the keyframes together, he has defined a variable which says the time for both animations to happen equals the time for the first plus the time for the second
        - -> so to do this in that section of code, it's timsed the variable in that sum by two (in other words, the time for the individual animation is the time, so the extra time it

adds to the total animation is produced from a delay)
        - -> he hadn't changed the second or first animations themselves, just the way that they are combined
    - *the final stage of the problem solving thought process*
        - -> this is to check the behaviour of the animation in the browser
        - -> he's thinking about it in terms of the impact that the UI it creates has on the user
- **Variety is the spice of life**
    - ***what this chapter section is about <- x(t) curves for keyframes***
        - -> this chapter section is on how to add timing functions (x(t) functions) to animations which are created using keyframes in Sass (indented css) <- timing functions had only been previously looked at for animations created using transitions
        - -> keyframes are made of many shorter states being placed together
    - ***the concept behind how this is done <- taking how this done for a single transition and then extending the concept to an animation created from combining multiple keyframes together***
        - *-> normally, for one transition you add in a cubic-bezier function*
            - -> this is a cubic x(t) curve for the speed of the transition
            - -> e.g the element being transitioned has a default cubic-bezier x(t) curve which makes it accelerate at first and then decelerate <- an ease-in-ease-out x(t) curve
            - -> you can choose the specific x(t) curve which the animation follows by using an online calculator for the cubic-bezier function
            - -> you can control the shape of the x(t) curve you want the animation to follow, and then the calculator will output four values -> which are the coordinates for two points on that graph
            - -> you plug those numbers into the x(t) curve feature for the animation which you set in Sass, the browser runs it and then the behaviour of the element being animated follows the x(t) curve which you created using the cubic-bezier online graphing calculator
            - -> that is the process normally used to control the x(t) curve of an animated element transitioning between two states
        - *-> to combine that method to keyframed animations*
            - -> keyframes animations are made of many smaller states which are all being transitioned together in one animation
            - -> you can add one x(t) curve which is default that all of these keyframes use
            - -> or, you can add different x(t) curves to some / all of the keyframes by themselves
            - -> the default x(t) curve for the transitions between the keyframes is the same as with a single transition (ease-in-ease-out)
            - -> if you wanted to target the x(t) curve for a single keyframe, you would do this using the same online graphic calculator method as you would for a single transition
                - this method is described above
    - ***an example which codes this concept using a loading bar creating using keyframes in Sass (indented css) <- this example changes the default x(t) curve which all of the keyframes use***
        - -> a) there is longhand for it and b) there is shorthand
        - a) longhand
            - $prog-bar-dur: 1000ms; <- *the variables are defined in a section at the top*
            - $prog-bar-delay: 150ms;
            - $cat-delay: $prog-bar-dur + $prog-bar-delay*2;
            -
            - .progress {
            - &__bar {
            - transform-origin: left;

- transform: scaleX(0.5);
- animation: progress-bar $prog-bar-dur $prog-bar-delay both;<- *the meaning of this was explored in the previous examples in these notes*
- ***animation-timing-function: cubic-bezier(.9,0,.1,1); <- this is the line which adds in the x(t) curve for the keyframes (all of them)***
- }
- }

- b) shorthand
  - $prog-bar-dur: 1000ms;
  - $prog-bar-delay: 150ms;
  - $cat-delay: $prog-bar-dur + $prog-bar-delay*2;
  -
  - .progress {
  - &__bar {
  - transform-origin: left;
  - transform: scaleX(0.5);
  - ***animation: progress-bar $prog-bar-dur $prog-bar-delay both cubic-bezier(.9,0,.1,1); <- then this is that same line in shorthand notation -> this has been added after the term both. In other words, before and after the animation happens, use the same styling as when it starts and after it finishes (don't revert it back to its origional value). The, during the transitions which make up the keyframe, use this x(t) curve for each of them. It's not the default ease-in-ease-out curve, it's one which he has defined using the graphic calculator method -> and that same curve is applying to each of the keyframes (this is how you change the default x(t) curve for each of those keyframes)***
  - }
  - }

- ***then we have the same example, but which changes the x(t) curve for one of they keyframes***
  - $prog-bar-dur: 1000ms;
  - $prog-bar-delay: 150ms;
  - $cat-delay: $prog-bar-dur + $prog-bar-delay*2;
  -
  - .progress {
  - &__bar {
  - transform-origin: left;
  - transform: scaleX(0.5);
  - animation: progress-bar $prog-bar-dur $prog-bar-delay both; <- *after 'both' in this line of code, this is where you would change the x(t) curve which all of the keyframes in the progress-bar keyframe are useing. There is no setting here - so it's reverting them back to the ease-in-ease-out x(t) curve for each keyframe, which is the default x(t) curve for transitions in Sass*
  - }
  - }
  -
  - @keyframes progress-bar{
  - 0% {
  - transform: scaleX(0);
  - opacity: .1;
  - }
  - 17% {
  - transform: scaleX(.18);

-
- }
- 24% {
- transform: scaleX(.40);
- animation-timing-function: cubic-bezier(.9,0,.1,1); <- *this line of Sass (indented css) is targeting one of the keyframes which make the animation. All of the other keyframes are using the same x(t) curve (which is the default ease-in-ease-out curve). Apart from this keyframe, which is using the x(t) curve with the cubic-bezier function whose profile can be found by plugging those numbers into its online graphing calculator*
- }
- 46% {
- transform: scaleX(.81);
- }
- 85%,100% {
- opacity: 1;
-
- }
- 100% {
- transform: scaleX(1);
- }
- }
- *further commentary*
  - -> he's then inspected the performance of the animation in the browser, and then gone back and changed the cubic-bezier function for two of the keyframes until he's found a combination which produces the result that wants
  - -> so we have the line before the keyframe is defined, which animates it
  - -> and then we have the definition of they keyframes themselves
  - -> this example was all about the use of the animation-timing-function property
    - this can be used in cases where more than one keyframes are defined and which adds delays between them
    - -> the other property which this can be combined with in Sass (indented css) is the fill-mode properties
    - -> the next chapter is about how to combined these approaches with more complex Sass (indented css)
      - -> to take multiples of what we have just done and make them more re-purposeable

- ***Let*'s recap!**
  - -> the start of animations which have been created using keyframes in Sass can be triggered by the pseudo-state of an element as with animations created using transitions
    - -> or which the case of animations created using keyframes, the animation can also be triggered by events which don't require user interactions (e.g the reloading of the element on the page, or the end of another animation created using keyframes)
  - -> controlling the timing of animations created using Sass keyframes
    - -> their delays
      - -> the animation-delay property <- in ms or s's, similar to transitions
      - -> before the transition (backward), after the transition (forward), or before and after the transtion (both)
    - -> their x(t) curves
      - -> setting their x(t) curves <- animation-timing-function
        - you can set the default x(t) curve for all transitions within a keyframe by changing the code in the shorthand notation which controls the keyframe,

above it in the code
    - -> alt., you can change the x(t) curve for individual transitions, by targeting the desired keyframes in Sass (indented css)

3. Manipulate and reuse CSS animations

- **Video notes <- see section 5. for these**
- **Taking stock of our keyframes superpowers**
    - -> this chapter is about how you use Sass to loop through / pause / play / rewind an animation created using a set of keyframes
        - -> playing those keyframes backwards
        - -> pausing them
        - -> looping them
        - -> playing them at different speeds
        - -> rotating elements at different speeds
    - -> rather than just one animation which plays once, forwards, and then stops
- Re**visiting input validation - out with the old**
    - -> how I would explain what he's written for this section
        - he's taken the html and Sacc (indented css) for a previous example and is replacing the transitions it uses with keyframes
            - -> this is a completely different example to the previous chapter's loading bar example
            - -> the example which this chapter uses is
                - -> it's a search bar
                - -> we have the html for the search bar
                    - -> there are label and input elements which have been contained together in a div
                    - -> the type for the form has been set in that label as an email
                    - this html is below
                        -
                        -
                        -
                        - <label for="email-input">email</label>
                        - <input type="email" name=”email-input>
                        -
                        -
                        -
                - -> then we have the old Sass (indented css) for that example
                    - -> this was when a transition was being used to create an animation for that search bar <- in other words, when the user entered text which wasn't in the format of an email and clicked off the search bar, then it would turn red using a pseudo-class in Sass (css) to indicate that the entered text was an invalid email
                        - -> the type of text which was entered in the label tag in html was an email
                            - -> the Sass knows that the text that the input field wants is an email
                            - -> it uses the expected form of this text to be able to decide in the Sass (indented css) if the text which has been entered is a valid email form or not
                            - -> then if it is an invalid email and the search bar isn't in its focus mode, then it's styling it red
                        - -> this example was appropriated from a previous chapter, the Sass

(indented css) which was used for this is below
- $cd-txt: #6300a0;
- $cd-txt--invalid: #fff;
- $cd-danger: #b20a37;
-
- .form {
- &__group {
- & input {
- border: 2px solid $cd-box;
- border-radius: 100rem;
- color: $cd-txt;
- font-family: 'Montserrat', sans-serif;
- font-size: 2.5rem;
- outline: none;
- padding: .5rem 1.5rem;
- width: 100%;
- transition: background-color 500ms;
- &:focus {
- border: 2px solid $cd-txt;
- }
- &:not(:focus):invalid {
- background-color: $cd-danger;
- border: 2px solid $cd-danger;
- color: $cd-txt--invalid;
- }
- }
- }
- }

- -> he's then deleted the transition which brought that red styling in
- -> and this is the example which we're working from
- -> the aim is to add in a keyframe, rather than a transition - to bring in that animation

- *-> in short*
  - *-> he's taken an example from a previous chapter -> which isn't the loading bar code from the previous section*
  - *-> this code is for a search bar. When text is in this search bar which is not in the form of an email and the search bar isn't in its focus state, then the entire thing transitions to become red*
  - *-> he's taken out the code which transitions the colour of the search bar*
  - *-> this was created using the transition property in Sass (indented css) and instead the aim of this chapter is to achieve a similar effect using an animation created with the keyframe approach in Sass (rather than transition)*

- **Form validation - don't you shake your head at me**
  - ***what he's using code to do in ths section of the chapter***
    - -> it's using the same Sass (indented css) as in the previous section
    - -> he took an example he'd previously used (a search bar which when clicked off and containing text of the wrong format - i.e not an email, turned red using a pseudo-class in Sass)
    - -> he removed that transition and is now replacing it with a keyframe animation
    - -> the entire idea of this example is to iterate through that keyframe multiple times to repeat the animation it creates, without having to repeat the code that defines it
    - -> he's replacing the old transition on that example with an animation created using

keyframes
    - -> that animation makes the search bar turn red and shake from side to side <- under the same conditions it previously had when the transition property was being used to create that animation
- ***the code he uses in this section of the chapter / the approach he uses to write it***
    - *-> he first defines the animation using keyframes*
        - -> the aim of the animation is to make the search bar move position from LHS to RHS under the conditions that were previously described
        - -> the full Sass (indented css) that he's used to do this is below
            - $cd-danger: #b20a37; <- *we begin by defining variables which are used in the Sass for the keyframe. We set the values of these variables before defining the code the for the keyframes*
            - $cd-txt: #6300a0;
            - $shake-intensity: 2%;
            -
            - .form { <- *then we write the code for the element which we want the keyframe animation to apply to (the entire search bar)*
            - &__group {
            - & input {
            - &:active, &:focus {
            - border: 2px solid $cd-txt;
            - }
            - &:not(:focus):invalid { <- *note how the names of these elements were defined using the BEM selector notation. This is the line of Sass (indented css) which applies the keyframe animation under the condition that the search bar is not in its focus state and it contains text which is not an email -> the type of text was labelled as email in the html for this section*
            - color: white;
            - border: 2px solid $cd-danger;
            - background: $cd-danger;
            - animation: headshake 100ms cubic-bezier(.4,.1,.6,.9); <- *this is the code which sets the animation under this condition. Headshake is the name f they keyframe below which this applies to , 100ms is its duration and cubic-bezier((insert numbers) the x(t) curve for the animaton*
            - }
            - }
            - }
            - }
            -
            - @keyframes headshake { <- *this is the code which defines the animation itself, by defininf a keyframe*
            - 25% { <- *for the first 25% of the animation, move the entire search bar to the left by the distance which is stored in the name of this variable, defined at the top of the page*
            - // far-right
            - transform: translateX($shake-intensity);
            - }
            - 75% {
            - // far-left
            - transform: translateX($shake-intensity * -1); <- *75% of the way into the animation, move the search bar by the same about it was moved to the right in the first 25% of the animation - move it that much, but in the opposite direction.*

After this, the animation returns the search bar back to its initial position
    - }
    - }
- ***then we have code which repeats the animation defined using keyframes -> we're iterating through it***
    - *where we were up to this point in the code*
        - -> so we have defined the animation using keyframes, which moves the search bar from right to left when there is text in it which isn't an email and there is no more text being typed in it
    - *the idea is - we want to iterate through (repeat) that keyframe animation as many times as we want*
        - -> then we want to repeat that animation as many times as specified, in order to optimise the behaviour of the effect -> in other words, to iterate through the animation
        - -> this is done by using the animation-iteration-count property
    - *to do this in the Sass (indented css)*
        - -> the line which was entered into the Sass (indented css) above to achieve this was **animation-iteration-count: 3;**
        - -> this line of Sass was entered in after the animation: headshake 100ms cubic-bezier(.4,.1,.6,.9); line in the code above
        - -> three iterations were used for the animation because there is a rule of thumb which calls it the rule of 3 <- it makes it look nice
- ***after adding in the effect, he puts in different numbers to see which combination makes them look better in the browser***
    - -> so, playing around with parameters like -> the distance that the search bar moves, the time it takes and the number of repeats (iterations) it goes through
    - -> when you write one keyframe (animation) and repeat it a certain number of times, it stops you from having to re-write the code which defined it each time you want to repeat the movement which the keyframes give in an animation
    - -> ***shorthand code for this is animation: headshake 100ms cubic-bezier(.4,.1,.6,.9) 2; <- the 2 at the end is the number of repeats (iterations) of the animation, aka keyframe***
    - -> when you have multiple repeats of the same animation, the duration which you enter in the Sass (indented css) is the time for one of the repeats of the animation (not all of them)

- **Note about the next two sections in this chapter**
    - **-> the example for the previous section of this chapter was on a search bar which lit up red and played an animation made using keyframes when its input wasn't an email and there was no text being entered in it**
    - **-> the example which they use for the last section of this chapter is for a set of loading bars**
        - **-> that example is split across several subheadings <- there are three of them**
            - **Loading animations - being put on hold**
            - **Looping animations - to infinity and beyond**
            - **Play state - playing and pausing animations**
        - **-> the first two are for making the animation in the example for the rest of this chapter <- the notes for these have been made together**
            - **-> the in the first one of those he's made the animation which plays once**
            - **-> and then in the second one, he's repeated it so it plays infinitely, duplicated it and then stacked the second one on the first one <- but the second one is a different colour and is playing in reverse order to the first**
                - **-> they are a set of animating search bars, he's made one of them in the**

previous section of the chapter
    - **-> then in these two sections**
        - **-> he's looped that one animation**
        - **-> then he's duplicated it and stacked the first one on top of it**
        - **-> then he's made the one on the bottom a darker shade of blue and animated in reverse to the one on the top -> so that they flow more easily together and aren't clashing as the animation goes on**
    - **-> the last subheading in these three is for adding in pause and play states to animations, which starts on that animation but is a different concept entirely**
- **-> so the notes for the rest of this chapter have been split into**
    - **-> the loading bar animation example <- "Loading animations - being put on hold" and "Looping animations - to infinity and beyond"**
    - **-> adding pause and play states to animations made using keyframes**
    - **-> the recap**

- **The loading bar animation example**
    - ***-> about this example***
        - This example is quite a long winded example split across two chapter sections called
            - **Loading animations - being put on hold** and
            - **Looping animations - to infinity and beyond**
        - *What the animation is*

            - -> these three images are pictures of the animation at three different points
            - -> in Sass (indented css) terms
                - *the elements which are being animated*
                    - *the idea behind one of the 5 sets of divs*
                        - -> these in this case are 5 divs on the top and 5 on the bottom
                        - -> they are two separate sets of divs -> the same just styled differently
                        - -> i.e, the 5 on the top and then the 5 on the bottom
                        - -> he first makes the 5 on the top, duplicates them and stacks another 5 below them
                        - -> but not just 5 -> he's written the code in a more generic way <- this is Sass with variables, so he's made the code which creates the divs in a way which allows you to state the number of divs that you want in each block of the loading bar
                        - -> you state the number of divs you want in each block of the loading bar from the variables in the Sass, and then it creates a block with that number of divs -> without you having to redefine one entirely
                        - -> that's the idea behind one of the 5 sets of divs
                    - *the idea of stacking them together*
                        - -> he has made one one those sections with 5 divs
                        - -> and then he has duplicated it and stacked it on top of the other section
                    - *then the idea of animating them*

- -> he's animated one of the sections with 5 divs before he's combined it with another section of 5 divs and then animated them together
- -> to animate one of the sections with 5 divs
  - he's iterated through each of the divs in the 5
  - -> added timing delays onto those transitions so they're staggered <- the timing delay when iterating through each of the 5 divs has been staggered by was timsed by the index of the div being iterated through (do the timing delay of the later divs was larger)
- -> the animation is translating the position of each of the bars in the y
- *-> then he's added in a time fill where*
  - for each of the sets of 5 divs the animation is running on a loop
  - he's set the loop for the elements on the bottom to run in reverse -> so that the bars on the top and bottom are in anti-phase
- *-> and then he's set them to run at the same time*
  - so the code which transitions one set of bars -> it's transitioning the first bar, then the second bar on a delay then the next slightly more delayed
  - -> for each set of 5 bars -> this is done through a for loop
  - -> and for each iteration of that for loop, the time delay that that specific bar has when it's being animated is increased by a factor of the index of that bar in the iteration
  - -> that's what happens for one block of 5 bars
  - -> and then in this example when they are adding in another block of 5 bars <- what they do is combine the for loops of each of those blocks together
  - -> then they take the bars in the bottom block, play the animation for them backwards and change their colour inn Sass to be a darker shade of blue

-> then in the next chapter they take that animation and start adding in pause and play states for it

- -> these two sections are about the creation of that animation in the first place

He goes through a long process to create the code for this example, which he details across the two chapters

- *-> the final code for this is below*
  - *this is the html for the example animation whose screenshots were given above in this note*

 <- *<u>each block of 5 divs was placed in its own div. The class for these divs was called "load" firstly because they were loading bars and secondly because you would play an animation like this in between the time when the styling for the page had loaded and when the server had returned data to fill the page for it with</u>*

 <- *<u>each of the bars for the loading divs were empty and only for the purpose of styling. Notice now each of them is named differently under BEM notation in Sass <- this is an example which had combined the two. Each of the bars could be formatted with its own class, but they were all instances of the larger class of loading bars</u>*

-
-
-
- <- *this div contains the 5 navy loading bars in the bottom of the animation, and was created by duplicating the code for the light blue ones in the top section of the animation*
- <- *the naming conventions used for this block of divs was the same as the set of the light blue divs, but they were instead referred to as -inv <- for inverse: the order which those bars were being animated in when they were iterated through was the same as for the top 5 divs, just backwards*
-
-
-
-
-
-

○ *this is the Sass*

- $cd-bars: #15DEA5; <- *moving the variables to the top of the page/ block of code*
- $cd-bars-inv: #0E397F;
- $size: 3vh;
- $anim-dur: 1000ms;
- $anim-delay: $anim-dur / 5; <- *taking the total time for all of the bars to move in one block of 5 when animated -> then dividing it by 5. This meant that the animations which affected the bars were being triggered at equal intervals in the transition. Normally, you might set all of the bars to transition at the same time (the beginning of the animation). This set the total length of the animation and then divided it into the number of bars - this was the time you wanted in between the start of the animation for one of the bars and the end of the animation for the other*
- .load {
- &__bar {
- animation: bars $anim-dur backwards infinite ease-in-out alternate; <- *this reads as: use the animation from the keyframe called bars, it's going to take this long per cycle of the animation, in the time between minus infinity and the event which triggers the start of the animation - style it the same as if it has just started, replay the thing infinitely (on a loop), use the x(t) curve for every bar in that animation as the ease-in-ease-out x(t) curve, how you loop through that animation is by running through it forwards and backwards -> alternate through that forever on a repeating loop*
- @for $i from 1 through 5 { <- *this is the code which iterates through each of the bars in the animation. We have two blocks of 5 bars, this is iterating through each of those 5 bars*
- &--#{$i} {
- animation-delay: $anim-delay * $i; <- *this iterates through the ith bar on the top of the animation*
- }
- &--#{$i}-inv { <- *this iterates through the ith bar on the bottom of the animation -> and they have been named accordingly using BEM selectors -> notice in the html for the bars on the bottom, the classes which each of the divs for those bars has is named differently (with an -inv) at the end of it*

- animation-delay: $anim-delay * $i; <- *what this is doing is setting the animation delay for each bar. We've taken the total time for the animation and divided by by the number of bars in that animation -> then that's the unit of time which we want each of the bars to be staggered by when the animation starts. So we're saying the time that the animation for that set of bars in the 5 which we are iterating through -> the delay at which that starts is that interval times the number bar which it is in / of the 5*
- animation-direction: alternate-reverse; <- *animations can play forwards or backwards. The animation which target the bars on the bottom is the same as the animation which targets the bars on the top. This is telling that animation to run backwards for the bars on the bottom, so that the two sets of bars will be in anti-phase. Not only is this in reverse, but when it's going from one loop of the animation to the next, on the one it's on it's playing it forwards, and then in the next step it's playing the same thing backwards rather than forwards (it's alternating)*
- background-color: $cd-bars-inv; *<- the colour of the bars on the bottom are different to the colour of the bars on the top*
- }
- }
- }
- }
- @keyframes bars { <- *you have the code which is defining the animation (this block), and then you have the code which is affecting how it's being used (the code above -> e.g in the classes for each of the individual blocks, being played forwards / forever / alternated between etc) <- that's how the animation is being used, This is how it's being created -> and in this case it's in Sass (indented css) using keyframes. The name of the keyframe for this animation is bars -> which is the same one being used above*
- 0% { <- *apply these styles to the elements being targeted when the animation is 0% fo the way through*
- transform: scaleY(0.5); <- *make the height of those elements equal to 50% of their default which was set elsewhere in the Sass -> this is a function*
- }
- 100% {
- transform: scaleY(1.0); <- *using the scaleY transformation function to set the height of the little divs (the loading bars) to take up 100% of the height of the div which they are contained in after the animation is complete (100% of the way through)*
- }
- }

○ *this is the css which it compiles into <- in other words the css which the Sass for loops produce, just without the for loops*

- .load__bar--1 {
- animation-delay: 200ms; <- *the compiled css is iterating through each of the bars -> these are the affects of the for loops -> the time delay for the animations of each of those methods has been calculated*
- }
- .load__bar--1-inv { <- *the order at which the css for these bars has been produced is the same as in the Sass -> the first bar on the top, then the first on the bottom and then the second on the top and then the second on the bottom -> and the naming conventions for these have been set up using*

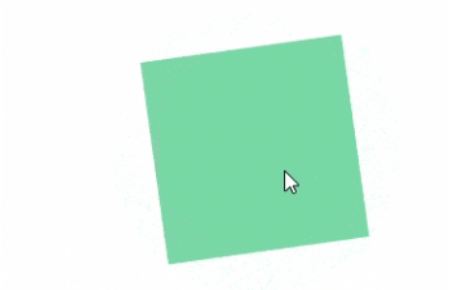
BEM selector notation. This automates the process of producing the css for those animations, and the Sass variables allow once change to be made which when compiled into css which affect everything

```
    animation-delay: 200ms;
    animation-direction: alternate-reverse;
    animation-fill-mode: forwards;
    background: #0E397F;
  }
  .load__bar--2 {
    animation-delay: 400ms;
  }
  .load__bar--2-inv {
    animation-delay: 400ms;
    animation-direction: alternate-reverse;
    animation-fill-mode: forwards;
    background: #0E397F;
  }
  .load__bar--3 {
    animation-delay: 600ms;
  }
  .load__bar--3-inv {
    animation-delay: 600ms;
    animation-direction: alternate-reverse;
    animation-fill-mode: forwards;
    background: #0E397F;
  }
  .load__bar--4 {
    animation-delay: 800ms;
  }
  .load__bar--4-inv {
    animation-delay: 800ms;
    animation-direction: alternate-reverse;
    animation-fill-mode: forwards;
    background: #0E397F;
  }
  .load__bar--5 {
    animation-delay: 1000ms;
  }
  .load__bar--5-inv {
    animation-delay: 1000ms;
    animation-direction: alternate-reverse;
    animation-fill-mode: forwards;
    background: #0E397F;
  }
```

- ***-> that is the content of those two sections of the chapter in a nutshell <- creating the loading bar animation***
  - ***to recap, these chapter sections were***
    - **Loading animations - being put on hold** and
    - **Looping animations - to infinity and beyond**
- ***-> the notes below are further commentary, which was mentioned within these sections themselves***
  - *-> the context under which an animation like this would be used / required*

    - the html and css et al, is loaded for a webpage
    - then it calls to the server for data to fill it
    - in time that the data takes to load onto the webpage, a loading animation like this can be played
    - it is true that the precise duration required for this animation is unknown, as the time that the server takes to retrieve its data is
    - this is why a looping animation is required, which continues to play in perpetuity
    - this animation is referred to as a placeholder <- something which is used while in the data for the content is being loaded onto the webpage
- *-> for this animation, there were two blocks which each contained 5 smaller divs. Those smaller divs (the ones being animated) were the child elements of the larger divs which contained them*
- *-> the entire point of staggering the animation for each of the little bars is for the UI / user engagement*
    - *when thinking about the different animations which are possible -> you can look at the functions in this course and then think about how you would combine them*
    - *-> the animation-delay property was used in this example to stagger the start of the animation across the different loading bars*
    - *-> the entire idea behind iterating through parts of code which are repetitive -> in the Sass (indented css) for the parts of the animation*
        - *-> repeating the block on the top and the block on the bottom*
        - *-> and then iterating through each of the little bars with a for loop*
- *-> the other axiom to this approach is looking at how the animation behaves after making a change to it, constantly - then interpreting that change and using how it behaves to inform the next set of changes which are made to the code*
- *-> there are also a certain set of standardised elements which he is going through and adding to it*
    - -> the fill mode (what the styling is before the animation starts - backward - and after it finishes - forward, or both)
    - -> the structure of animations made with keyframes is also generally similar
    - *-> there is the definition of the animation itself done using @keyframes, and then there is its use -> in the code which styles the elements it wants to target*
        - *-> the number of times that animation is looped (iterated) through*
    - *-> telling it to play infinitely, or the direction that it's being told to play in (backwards, normal, back and forth aka alternating), creating loops, the timing function x(t) for the animation*
        - -> in this example it's seen as an alternative to adding in new frames -> he's minimised it to two
        - -> which of the bars the timing (x(t)) function you define you want to be targeted <- in other words, what the timing function is and where you want it to be defined (in the code for the individual keyframes, or in the code which sets the use of the keyframe animation on an element of the webpage <- outside the definition of that keyframe in Sass itself, doing that will set the default timing function of the elements being animated - vs for a single keyframe)
            - -> when working out the specific timing functions to use, it's the same approach as with the cubic-beznier functions for transitions in this note above
        - -> some elements might need the animation played forwards on them and some elements might need it played backwards <- working out which direction you want the animation to go on for each of those elements on the webpage
            - -> and then (in this example) executing that with for loop(s)
- *-> another piece of language which commonly used <- iteration. Each time the animation*

is played, e.g on a repeating loop -> that is an iteration
    - *-> he writes out the code in longhand, gets it to be functional and then simplifies it down to be more concisely written in shorthand notation*
        - -> inverted sync <- technical name for this animation
            - secondary motion is another one
        - -> the reason why the formats of the names for the bottom 5 bars in the animation are different is because they are being animated in reverse <- using the animation-direction property in their Sass (indented css)
- **Adding pause and play states to animations made using keyframes -> "Play state - playing and pausing animations**
    - ***-> how I would summarise this section of the chapter***
        - so, this section is on pausing and playing animations made with keyframes in Sass (indented css)
        - ***he does this using two examples***
            - -> the first is a rotating box <- there is the sass and html for this, which I will include in these notes and annotate
            - -> the second is applying those concepts to the example from the previous sections of the chapter <- taking the loading bar animation and telling it to stop the bar which is being hovered over from moving, and that bar which the effect is for is one of the 5
                - I think that hover effect is only for the bottom set of 5 bars, but we'll investigate the Sass for this later
    - ***-> the first is the example for the rotating box***
        - *-> how this effect works is*
            - -> we have a box, this is an empty div which has been formatted in Sass
            - -> then we have the animation which is causing it to rotate
                - the animation for this example was defined using @keyframes
                - in these examples, there is always code which
                    - a) applies the animation to specific elements on the page and
                    - b) defines the animation itself
                - the code which is defining the @keyframe in the examples in this course comes after it's been used
                    - -> so in the Sass you are seeing the css for the box <- in there the animation is being applied
                    - -> then you are seeing the css which defines the animation which is rotating it
                    - -> this Sass which defines the animation using @keyframes is coming after the use of it for individual elements using the animation parameter



            - -> so at first, we have the Sass for the box
                - this is literally an html div which was styled into a box
                - -> we have the styling for it
                - -> and then (in the Sass which styles that box) we have Sass which applies the animation which was created later in the code using keyframes
                - -> that code is rotating the box
                - -> then in this example <- the entire purpose is to add a hover state into the rotating box
                    - when that box is being hovered over, then the animation which is rotating it pauses
                    - -> this was all defined under the code which applies the animation to that

box (rather than the definition of the animation itself)
        - *-> there is the Sass (indented css) which applies the animation to an element -> this is under the Sass for the elements where the animation is being applied*
            - *-> in that same code - for the elements where the animation is being applied <- that is where the animation state can be set to pause / play etc*
            - *-> this isn't done in the definition of the animation, it's done in the code which applies it to an element / under that element*
        - *-> then there is the Sass which defines the animation and how it behaves (this is, in this case created using @keyframes and is defined elsewhere in the code)*
        - *-> it's like having a dictionary which defines a word (@keyframes) and how it behaves / what it's meaning is, and then the word being used throughout the styling for the webpage on various elements*
- *-> this is the code for the rotating box example*
    - *the html*
        -
        - <- *the box is literally an empty div which has been styled into a square, this is the html for it -> empty div, class of "spin"*
        -
    - *the Sass*
        - .spin {
        - background-color: #15DEA5; <- *this is the Sass for the box itself - we start off with the Sass which turns the div into a square and gives it its colour*
        - width: 30vh;
        - height: 30vh;
        - animation: spin 3s linear infinite paused; *<- then we have the Sass which applied the animation to the box - this is rotating it. Spin is the name of the animation which was defined using the keyframe below, the duration is 3s - one spin every three seconds, a linear x(t) curve, the animation plays an infinite about of times and doesn't stop when it's running. The status of the animation is paused -> until it's played - at which point it runs an infinite of amount of times*
        - &:hover {
        - animation-play-state: running; <- *this is the code which (inside the Sass for the box) plays the animation when the box is being hovered over. So, the default state which was set for it above is that the animation is paused - this plays the animation on the box when it's being hovered over. The effect is for the box to not do anything while it's being left alone - and then when it's being hovered over it's rotating (the animation is playing -> 'running'). The line which is being used is 'animation-play-state' -> and an example of the shorthand for this line of code is in the example above (where it says 'pause') -> and then this line of code is its syntax in longhand*
        - }
        - }
        -
        - @keyframes spin { <- *typically we see the variables which are used within the Sass defined in the code at the top of the example. And then there is the Sass for the application of those animations to individual elements on the page. This is the code for the definition of the animation itself, which comes after when it has been used in the code. This animation was created using the keyframes method and the set of these keyframes is called spin*

-
- from { <- *from is another word for 0% (apply this Sass, indented css from the start of when the element loads on the page)*
- transform: rotate(0deg); <- *in this period, start rotating the box from an upright position*
- }
- to { <- *the from and to keywords in keyframes apply for cases where there are two states being transitioned between. We are dealing with two transition states -> that is when from and to are used, and to is the final transition state of the box / keyframe which we want it to animate through*
- transform: rotate(360deg); <- *end the animation where the box started, one rotation later*
- }
- }

- *-> commentary for the rotating box example*
  - *about*
    - -> the notes above this section have explained what the code he produced for this example did and the context
    - -> this commentary includes extra points which he mentioned in the course material for the box example
  - *commentary*
    - -> an example use case for this is where the data for the page hasn't loaded yet and we are making an interactive game for the user
      - -> the animation is a part of the game / is it
    - -> it's the animation play state property which is being used to do this
      - -> pause
    - -> we are setting the animation to run infinitely in the code where it has been applied to the element in the Sass (indented css)
      - -> you can also set x(t) curves here <- linear in this example
    - -> we are rotating the box by 360 degrees
    - -> this is easy to overdo and has to be done in moderation

- ***the second example (of applying the pause state in Sass animations created using @keyframes) is going back to the loading bars from the previous section of the chapter***
  - *aim / context of this example*
    - -> the aim is to apply the same 'pause the animation when the mouse is hovering over this element' nomenclature as in the previous rotating box example -> but to the loading bar animation example from the previous section of the chapter
  - *this was again gone with the animation-play-state*
    - -> to take the animation created using Sass keyframes and pause it when it was being hovered over
    - -> being hovered over <- in other words, a pseudo-state in Sass
      - -> user interactions
    - -> the alibi for this example is that we are making the loading bar animation more interactive to act as a game for the users in between the time when the page loads and the data does <- aka when the loading bars appear
  - *there are two modifications to the code from this*
    - -> so he's taken the Sass for the loading bar animation example and then added two alterations to the code for that
      - -> the first alteration was to randomise the time delay in between the start of the animations for the different bars
      - -> the second alteration was to add in code with a similar effect to the spinning box <- hover over one of those bars and the animation pauses

- -> normally the time delay would be
  - -> for one of the blocks of 5 bars in them -> he took the total time for that animation to run and divided it by the number of bars
  - -> then he delayed the start time for the animation of each of those bars
  - -> he did this by iterating through each of them
  - -> the start time of the animation running on the 4th bar e.g would be the total time for the animation on 5 bars, over 5, then all timsed by 4
  - -> the time delay in between the animation for each of those bars was defined in a way so that the animations for them would be spaced evenly
  - -> what he's doing in the first alteration to that code -> is to randomise the start times of the animations for the 5 little bars -> the navy blue ones in the bottom of the animation
- -> then in the second alteration to the code
  - -> he's taking the alteration which was made to the code for the box example
    - -> hover over the box and it the animation starts
    - -> the animation was set to run for the box -> but it's default state was paused
    - -> and then in that same section of Sass (under the code for the spinning box itself), and after where the animation had been applied to it with its default state as paused
      - -> this was where the hover state of the box was then defined
      - -> and the play-state of the animation for the box was set to active
      - -> so the box would start moving when it was being hovered over
      - -> and then at the end of the Sass for that example, that's where the animation was being defined in the @keyframes
      - -> if there were variables used in that Sass, they would have been defined at the top of the code for easy access then used throughout it
    - -> so back to the loading bars example
      - -> the second alteration <- getting the animation on one of the bars to pause when it's being hovered over
      - -> the box example was getting the animation to play when it was being hovered over, this is the same effect in reverse
      - -> hovering over one of the little navy bars on the bottom -> then the animation for it stops
      - -> and because the animation time was started randomly -> the aim is for the user to then play with it -> to hover over it such that it pauses the animation in time for it to be in anti-phase with its partner bar on the top
      - -> that entire thing is set to be in the time between when the page loads and its data has been retrieved from its server -> as a game that the user can play with the loading bar animation while they wait for the data to load onto the page

- *-> the code for this*
  - *the html*
    -
    - Hover over the blue bars to synchronise them *<u><- the only change to the html in this case is that he's added in text below and above the loading bars in order to explain the 'game' to the user which is now added by the Sass</u>*
    -
    -
    -

-
-
-
-
-
-
-
-
-
-
-
- with their mint-green partners above.
-

- *the Sass*
  - $cd-bars: #15DEA5; *<- variables at the top*
  - $cd-bars-inv: #0E397F;
  - $size: 3vh;
  - $anim-dur: 1000ms;
  - $anim-delay: $anim-dur / 5;
  -
  - .load { *<- then the Sass for the element which the animation is being applied to (this isn't the definition of the animation itself, that's defined in @keyframes at the bottom of the Sass -> this is the use of it)*
  - &__bar {
  - animation: bars $anim-dur backwards infinite ease-in-out alternate;
  - @for $i from 1 through 5 {
  - &--#{$i} {
  - animation-delay: $anim-delay * $i;
  - }
  - &--#{$i}-inv {
  - animation-delay: $anim-delay * $i + random(100)*15ms; *<- this is the first alteration to the Sass for the loading bars -> the start times for the animations on each of the navy loading bars at the bottom of the animation are now random. These have been randomised so that when the user pauses a navy loading bar in its hover state, they can stop hovering over it when in anti-phase with its light blue counterpart. This hover effect which pauses the animations for these elements and the randomisation of their respective animation start times is done on a bar by bar basis for the block of navy bars at the bottom of the animation, while the Sass for the light blue bars at the top of the animation remains the same as with the code from which this example was originally appropriated*
  - animation-direction: alternate-reverse;
  - animation-fill-mode: forwards;
  - background-color: $cd-bars-inv;
  - &:hover {
  - animation-play-state: paused; *<- this is the Sass which, when the respective navy loading bar is in its hover state -> its animation now pauses. This is the same approach as with the rotating box in the previous example, except the animation the hover state effects is pausing it in this example - as opposed to playing it and its default being paused*
  - }
  - }

        - }
        - }
        - }
    - *-> further commentary <- notes on other points he mentions in this section of the chapter*
        - *<u>-> the next chapter is exploring the Chrome DevTools for animations made using @keyframes (the method for creating Sass animations this was explored for in previous chapters was with transitions)</u>*
        - -> the entire context of this is to create a game which the user can play when the data is loading onto the page
            - <u>-> the Sass random function</u>, in order to randomise the start times of the animation for the loading bar
            - <u>-> the argument for this function was the range the random number was required to lie between</u>
                - <u>-> and the lowest possible number if not otherwise stated was 1</u>
        - -> x(t) transition functions <- in the box example it was linear
        - -> he's also edited the html -> with text for instructions targeted at the user when interacting with the loading game

- **Let's recap!**
    - *Sass to change the direction and number of times an animation of played*
        - ***parameters which you can put into the animation-iteration-count property***
            - <u>animation-iteration-count: n;</u> <- to repeat an animation n times
                - e.g animation-iteration-count: 6; //this replays the animation six times
            - <u>animation-iteration-count: infinite</u>; <- to play an animation for perpetuity
                - e.g animation-iteration-count: infinite; //this replays the animation forever
        - ***parameters which you can put into the animation-direction property***
            - *<u>Animation-direction: normal</u>;* //to play back the animation normally from beginning to end (this line of Sass in an actual project could be redundant in most cases)
            - *<u>Animation-direction: reverse;</u>* //this will play the animation in reverse (from end to beginning)
            - *<u>Animation-direction: alternate</u>;* //to play the animation from beginning to end and then from end to beginning
            - *<u>Animation-direction: alternate-reverse</u>;* //to the play the animation from its end to beginning, then from its beginning to end <- basically backwards and then forwards when it's done
        - ***parameters which you can put into the animation-play-state property***
            - *<u>Animation-play-state: paused;</u>* <- to pause an animation which would otherwise play (this can be used when the animation is applied, and then removed for its hover state - or vice versa)
            - *<u>Animation-play-state: running;</u>* <- this is the same as the pause state, but it instead plays the animation -> like a remote control, this is the code which resumes the animation if it's been paused
                - -> the animation for this has to be made using an @keyframe directive in Sass

4. Refine your animations with Chrome's animation tool *<u><- notes for this chapter have been written in their own file, called '8iv Create Modern CSS Animations'</u>*

- **Video notes <- see section 5 for these**
- **Where's my palette and easel?**
- **DevTools - demystifying animations**
- **Adjusting animations - slip sliding away**
- **Assessing animations - that’s just, like, your opinion, man**
- **Post mortem - to each their own**

- **Let's recap!**

5. Put it all together

- **Video notes**
  - ***-> about this section***
    - each chapter of the course has a different video
    - -> these are notes on the videos for chapters 2, 3, 4 and 5
    - -> these aren't comprehensive in the sense that they don't cover all of the course material
    - -> these are brief introductory style videos for the content of each of these chapters
    - -> the content was in depth text which was to be read through (this document is notes on those readings)
    - -> and this section, notes on the introductory videos for the content at the beginning of those chapters
  - **2. Construct CSS animations using the CSS animation properties**
    - -> keyframes vs transitions <- both can be used to make animations
    - -> this chapter was about timing functions on individual keyframes, and the difference between keyframes and transitions when creating animations using Sass
    - -> transitions only exist within the selectors where they've been declared
    - -> keyframes are available globally -> so any selector can use them
      - -> they are based off of percentages rather than durations
      - -> they can have different durations and delays
  - **3. Manipulate and reuse CSS animations**
    - -> manipulating and reusing CSS animations
    - -> keyframes can be reused to create animations, and those animations can be replayed backwards
    - -> they can be played forwards / backwards / in loops / certain numbers of times / pausing them <- this chapter was about how this was done in animations created using those keyframes
  - **4. Refine your animations with Chrome's animation tool**
    - -> animations which are too long / which need to start later
    - -> when we watch them after we've first created them
    - -> we judge how long the animation should take based off of what subjectively feels correct
    - -> this chapter is on using the Chrome DevTools for animations which have been created using @keyframes in Sass
    - -> you could find the right the parameters for an animation made using @keyframes in Sass by changing their values in the IDE and then checking the appearance of the animation it gives in the browser -> you can also do it using Chrome's animation tools
  - **5. Put it all together**
    - -> principles of animations <- the benefits of adding them to a page
    - -> properties of transitions
      - -> transform functions <- x(t)
      - -> colour transitions via the opacity property and pseudo-elements
    - -> multi-stage transformations
      - -> looping animations
      - -> iterations
      - -> direction properties
    - -> the peer to peer activity at the end of the course
      - -> practicing the skills from this course
      - -> peer feedback on animations
    - -> this course <- Sass (indented css) animations

        - -> graphic effects
- **You've done a beautiful thing**
    - *these are notes on the text below the video in the last section of the course, which is a recap section*
        - -> Sass / css animations
        - -> practicing / portfolio sites
        - -> the 12 principles of animations
        - *approaches used to create animations in Sass*
            - -> css transitions
            - -> @keyframes
        - *approaches used to alter animations in Sass*
            - -> looping animations using the iteration and direction properties
        - -> the entire course was on animations in Sass (indented css) -> which were created using transitions in the first two thirds of the course, and then using @keyframes in the final third <- including an in depth exploration of how these could be looped through etc