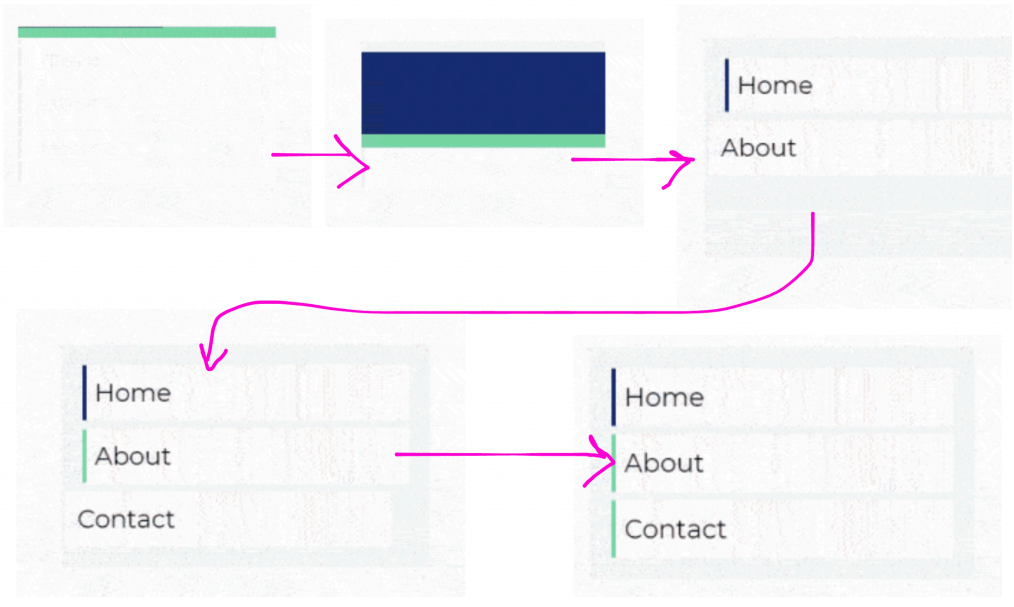
4. Refine your animations with Chrome's animation tool

- **Video notes <- see section 5 for these**
- ***How I would outline the rest of the content for this section***
  - ***The main concept behind this chapter***
    - -> an animation is like a video <- think of making a video in iMovie
      - there are different layers to it
      - each element in the animation doing something as time goes on -> is like an layer in each of those streams on iMovie
        - -> except in this case the animation was made using @keyframes in Sass (indented css)
      - -> the process which this entire chapter goes through (apart from the recap) is how to optimise the times for each of those sections in the iMovie <- i.e their durations, delays
        - -> so, when they start relative to each other and how long they're taking
        - -> each of the elements which is being animated is like a layer in that stream
        - -> you can see the x(t) curves which they are following in those streams
        - -> the shape of those x(t) curves in the streams takes the form of the cubic-beznier curve which was defined in the Sass
      - -> the entire idea is that once an animation has been made
        - -> it's a combination of many different elements, and each of them are being animated
        - -> this 'iMovie' process is done by inspecting the animation using the Chrome DevTools
        - -> each of those elements is broken down into a different layer
        - -> what we are doing then - once the animation has been created in Sass - is opening it in the Chrome animation DevTools (aka like iMovie) and playing around with the durations and time delays of each of those elements in the animation
        - -> we are doing this in the DOM (in the browser, like the console) without the code in the IDE -> to play around with those timings relative to each other to optimise them
        - -> then once we have the timings which we like
          - -> we the DOM tells us the numbers for them
            - -> it's telling us the changes which were made in comparison to the source code
            - -> it's like opening the animation in an iMovie, playing around with with the duration and delays of each of the elements relative to each other
            - -> then exporting the numbers which enable that to happen
              - ->the DevTools are telling us what the difference in between the code that we have produced and the source code for the animation is <- this is called diffing
              - -> and then we are exporting the numbers for the changed elements and then manually inserting them into the source code into the IDE so that the elements in the source code and the modified ones in the DOM match
  - ***An outline of the structure for this chapter***
    - *-> the first the first section of this chapter is about how the animation Chrome DevTools are used to inspect an animation created with the Sass @keyframes feature*
      - *-> the UI / features of these Chrome animation inspection tools*
    - *-> the chapter then explores the use of these tools on an example animation, to optimise the duration and start times of its respective elements relative to each other*
      - *-> each element in this animation which is having its behaviour altered looks like a*

layer in an iMovie stream
        - *-> with the x(t) curves visible for its respective elements*
        - *-> tweaking the start / end times / durations for each of those elements on the stream*
        - *-> and then having the Chrome DevTools give the numbers for the start times / durations of those optimised animations*
        - *-> then enter then back into the Sass which originally produced them*
        - *-> the example animation for this is*
            - *a larger div, like a menu bar*
            - *that contains three smaller ones, which animate in from left to right*
    - *-> we then have a recap of the key concepts at the end*
- ***How the notes for this chapter are organised***
    - -> I am starting off the notes for this chapter with an exploration the final code for the example which is produced in the second half of this chapter
    - -> then it within the context of Chrome's animation DevTools
        - in other words, iMovie for animations made using Sass
        - -> the UI of those tools
        - -> and then the approach used when applying them to the example animation in the first half of the notes for this chapter
            - to optimise the duration / start / ending times for the Sass
    - this is how the sections for this chapter were organised in this course
        - 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
        - -> and then there is the recap which these notes will go over at the end of the content for the chapter
- ***The code for the example in the second half of this course***
    - *commentary*
        - -> we first begin with the Sass / html for the example which was used in the second half of the content for the chapter
            - -> there is one example animation which is being optimised in this chapter
        - -> the important parts to note for this are that the various sections which are being animated are later inspected in the Chrome DevTools animation suite, and each of them has their own timeline in that stream <- like iMovie
    - *the appearance of the animation which this code produces*



- ***BELOW IS THE HTML AND SASS FOR THE EXAMPLE ANIMATION WHICH THIS CHAPTER USES, AND ANNOTATIONS PRODUCING A COMMENTARY ABOUT HOW IT PRODUCES THE EFFECT IN THE SERIES OF IMAGES ABOVE***
  - ***-> the html (for the animation optimised using Chrome DevTools this chapter)***
    - <- *this is the div around the entire menu bar being animated*
    - <menu class="menu"> <- *we are using the semantic <menu> html tags*
    - <- *the little div which is inside the larger div -> there are three of these, and this is the first - in this case for the home element on that menu bar. Notice how the classes which have been used to target those elements follow the naming conventions of BEM notation*
    - Home
    -
    -
    - *<- then we have the second little div inside the big div, this is for the About section the menu bar*
    - About
    -
    -
    - <- *likewise, the third little div in the menu bar -> for the contact section*
    - Contact
    -
    -
    - <- *these are the two empty divs which transition into the menu bar at the beginning of the animation -> notice how the names of the classes which targets them have been named accordingly -> the second one is for the mint rectangle and the first is for the navy one*
    -
    - </menu>
    -
  - ***-> the Sass (for the animation optimised using Chrome DevTools this chapter)***
    - $cd-navy: #0E397F; <- ***the variables which are used in the Sass are always defined at the top of the code -> before they are used***
    - $cd-mint: #15dea5;
    -
    - @mixin menu__open-accent($dur, $delay) { <- ***then we have a mixin defined -> in other words, the equivalent of a function in Sass which targets the styling of certain elements on the page. css is a scripting language*** *-> if you were to have a styles.css file then the structure of the code would be less object-oriented. This is what Sass does <- a mixin is the equivalent of a function in Sass which changes the styling of various elements, when it it called in the rest of the Sass using the @include include keyword. This is the definition of that function -> and then later in the Sass it will be used by calling it using the line @include menu-open-accent. $dur and $delay are the arguments which this function take. Open accent targets the mint and navy rectangles which transition into the animation as it loads*
    - content: "";
    - position: absolute;
    - left: 0;
    - right: 0;
    - top: 0;
    - bottom: 0;
    - background: #f4f9f8;

- transform-origin: top left; <- *this is the animation which transitions the mint and navy rectangles from the left to the right of the page. The start of this animation is on the top left of the menu div which contains them <- this line of code is literally taking the origin where that animation starts from and moving it to the top left hand corner of that div*
- animation: menu $dur $delay both; <- *then this is the line of Sass which dictates the use of the animation for the two accent divs which move in at its start. This is a keyframe which will later be defined in the code, after its use. This line is shorthand notation, which then sets the duration of that initial accent animation, combined with the time in between the loading of the menu on the page and the start of the animation, and the fill option which is telling it -> the styles which you apply in between minus infinity and when the animation starts are the same as the instance where it starts -> and the styles which you apply in between when the animation ends and plus infinity are the same as when it ends (both)*
- z-index: -1; <- *this line is telling the browser to stack those two accent divs behind the larger div for the menu bar which contain them -> this animation is combining multiple animations (the first is the one which this mixin was defined for) -> when it's over, hide the two divs which were being introduced back behind the menu bar - while the rest of the elements on it can be animated*
- }
-
- .menu { ***<- then we have the Sass which styles the elements on the webpage which are being animated -> this is also the section of the code where the use of those animations has been set, relative to each other. The last section of the code is where these animations have been defined using @keyframes -> this section contains the instances which describe how they have been used for those specific elements which are being animated***
- min-width: 33vh; *<- this is the Sass (below) which orders the menu items by flexing them into a column with space in between them*
- display: flex;
- flex-direction: column;
- justify-content: space-between;
- padding: 1rem;
- background:$cd-mint;
- overflow: hidden;
- transform-origin: top left; <- *when those menu items are being animated in, the animation starts from the top left hand side of the screen. This line of code moves the origin (centre) of that animation - to the left of the menu div which contains them - and at the top*
- position: relative;
- animation: menu 661ms both; <- ***after the code which defines the appearance of the elements when the animation is not moving -> we then have the code which applies the animation to those elements (this is before the code which defines the the animations themselves, this comes last with keyframes)***. *This line of code is shorthand notation for -> animate the menu bar items with the animation which was defined with the keyframe called menu, have it take 661ms and it uses the both keyword -> this is the same which was used above (see the annotations around this for the sake of brevity)*
- z-index: -10; <- *the two classes / selectors which have been defined below in Sass are for the two 'open accents' (the divs for the navy and mint rectangles in the code)*
- &__open-accent--1{

- @include menu__open-accent(450ms, 275ms); <- *this first selector is applying the styling function (Sass mixin) which was defined above to the mint div / rectangle which is being animated in before the menu items themselves ->* ***notice how the order these selectors have been defined in this section of the Sass is the same as the order which they appear in the animation***
- }
- &__open-accent--2 {
- @include menu__open-accent(450ms, 150ms); <- *then we have the Sass for the first nacy rectangle which animates in, this line is applying the Sass mixin on it - similarly to its mint counterpart being targeted in the Sass selector above*
- background: $cd-navy;
- z-index: -2; <- *moving it behind the menu div in the stack it is being placed on*
- }
- &__item { <- *then we begin the Sass which applies the animations defined using the mixin (below this section) to each of the menu items - there are three of them in this case, and this is the general class which is being defined for them*
- padding: .75rem; <- *formatting them*
- margin: .25rem;
- background: #fff;
- animation: menu__item 810ms cubic-bezier(.1,.9,.1,1) both; <- *applying the animation to them - this is the same shorthand notation as with the examples above. The main difference is that we have an x(t) curve -> as with the previous notes in this course, this cubic-beznier equation could be derived from an online graphing calculator - the first and last two values in its sequence corresponding to two points on that curve*
- position: relative;
- &-accent { <- *then we have the code which styles each of the elements of those individual menu items while they are being animated -> this is the selector for the smaller divs to their left which are being animated in. Notice how their naming convention is produced according to the styling used by the BEM convention*
- content: ""; <- *since those elements for the accents on each individual menu item were 1) created using divs 2) targeted in the Sass with animations made using @keyframes and 3) had no content -> if this line was omitted then the animations on them wouldn't have worked*
- position: absolute;
- top: 0;
- bottom: 0;
- left: 0;
- width: .25rem;
- background-color: $cd-mint;
- animation: menu__accent 400ms both; <- *then (after we have the code which formats them), we have the the shorthand Sass for their animations. This is the same as the shorthand in the Sass above -> and the name of the @keyframe which defines the animation this time is menu_accent*
- &--active {
- background-color: $cd-navy <- *syntax, there is no semi-colon at the end of this line because the code which was defined for this element came from a variable -> in the active state of the little menu bars*
- }
- }
- &--1 {

- animation-delay: 475ms; <- *each of the little menu items have been targeted in Sass with an animation which moves them from left to right when they enter in. The next three BEM selectors are setting time delays to the divs around these little menu items, so that they stagger in. This top line controls the time delay in between the end of the animation for the two accent divs (navy and light green) and the start times for the animations which move the menu items onto the page. Those menu items have two parts -> the menu items themselves and then the accent divs on them. The time delay for the first div defined in that was set using the Sass in this line, and the time delay for the decorative div in that was defined using the Sass in the line below*
- > div {
- animation-delay: 712ms;
- }
- }
- &--2 { <- *the code for the next two selectors is the same as in the previous one but with alternate content. Each of these corresponds to a menu item (there are three)*
- animation-delay: 546ms;
- > div {
- animation-delay: 805ms;
- }
- }
- &--3 {
- animation-delay: 632ms;
- > div {
- animation-delay: 914ms;
- }
- }
- }
- }
-
- @keyframes menu { <- ***then at the end of the code, we have the keyframes which define how the animations behave. It's like with a Python function, but in reverse. You have the code which defines how it works, and then you have the code which calls the function in various cases where it's being applied. So you have the code which defines it, and the code which uses it. With animations made using keyframes - the code which defines how that keyframe works is written at the end of the document. In the code above, we've seen how the keyframe which is defined here was used. Every time the Sass said 'animation: menu__accent 400ms both;', for example <- the keyframe with the name menu_accent was being called / applied. This code at the end of the Sass is defining how that keyframe itself works. It's name is menu - we are reaching the end of the code, and in its last section - this is where all of the Sass which defines the keyframes used above was set***
- 0% { <- *from t=0% to t=33% of the way through the animation / transition*
- transform: scale(0,.07); <- *do this to the content which it's being applied to -> expand its size by a sf of 0 in the x, and of 0.07 in the y*
- }
- 33% { <- *from t=33% of the way through the animation to t=100% of the way through the animation*
- transform: scale(1,.07); <- *do this to the content which it's being applied to -> expand its size by a sf of 1 in the x, and of 0.07 in the y*

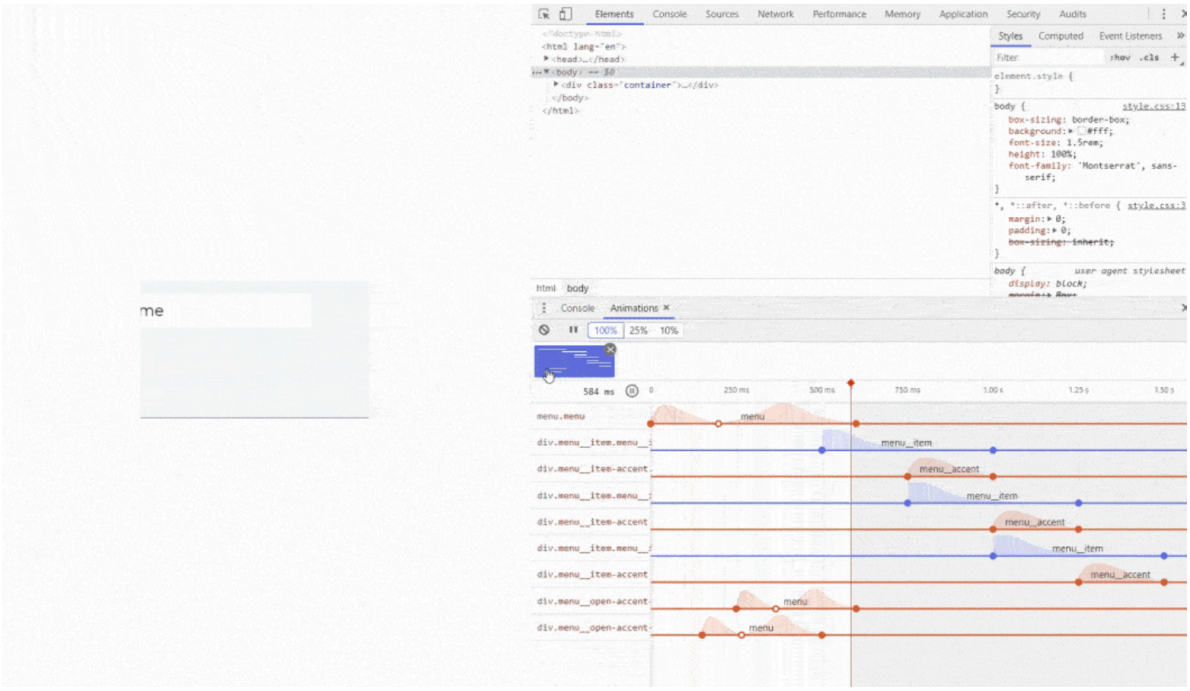
- animation-timing-function: cubic-bezier(.73,.01,.2,.99) <- *when you apply the animation to the content using this keyframe, make its x(t) curve follow this equation -> these numbers came from an online graphing calculator for the cubic-beznier function and the shape of this x(t) curve can be seen later in the chapter when the Chrome DevTools are generating its shape*
- }
-
- }
-
- @keyframes menu__item { <- *define the keyframe called menu_item*
- 0% { <- *from 0% of the way through animation to 100% of the way through the animation -> in other words, the styles defined under this section of the keyframe are working for the entire duration of the animation which it defines*
- transform: translateX(-110%); <- *apply these styles, in other words -> move the element which it's targeting to the left by 110% of its width*
- }
- }
-
- @keyframes menu__accent { <- *define the keyframe called menu_accent*
- 0% { <- *from 0% of the way through animation to 100% of the way through the animation -> in other words, the styles defined under this section of the keyframe are working for the entire duration of the animation which it defines*
- transform: scaleY(0); <- *make the element which it's targeting have a starting height of 0. By the end of the animation, transition this height to the value which was defined as its default in the rest of the Sass. This keyframe was used to target the accent divs on the left of each of the three menu items -> initialising the heights of these accent divs to 0 at the start of the animation and then transitioning them to 100% by its end*
- }
- }

- ***The code for the example in the second half of this course***
  - *-> commentary about where we are in the context of this chapter*
    - -> that was the html and Sass for the example animation which this chapter uses
    - -> there are multiple elements which are being animated and multiple animations which have been used to target them
      - *the sequence that these animations on these elements followed was (this is a long winded timeline of them)*
        - we start off with an empty div for a menu, and then introduce two smaller accents divs to this (a mint one and a navy one)
        - -> then we get rid of the accent divs by animating them out
        - -> then we fill the empty space that this leaves on the menu div with three smaller menu items
        - -> those menu items are animated in using the same animation -> but there is a delay which has been applied to the ones on the bottom
        - -> then those divs for the menu bar themselves have smaller accent divs on them
        - -> those accent divs are being targeted with an animation which was defined using keyframes
          - -> their heights are starting at 0, and then by the time the menu which they have been placed on transitions in -> then their heights take up 100% of the smaller menu div which they have been placed on, of which there are three

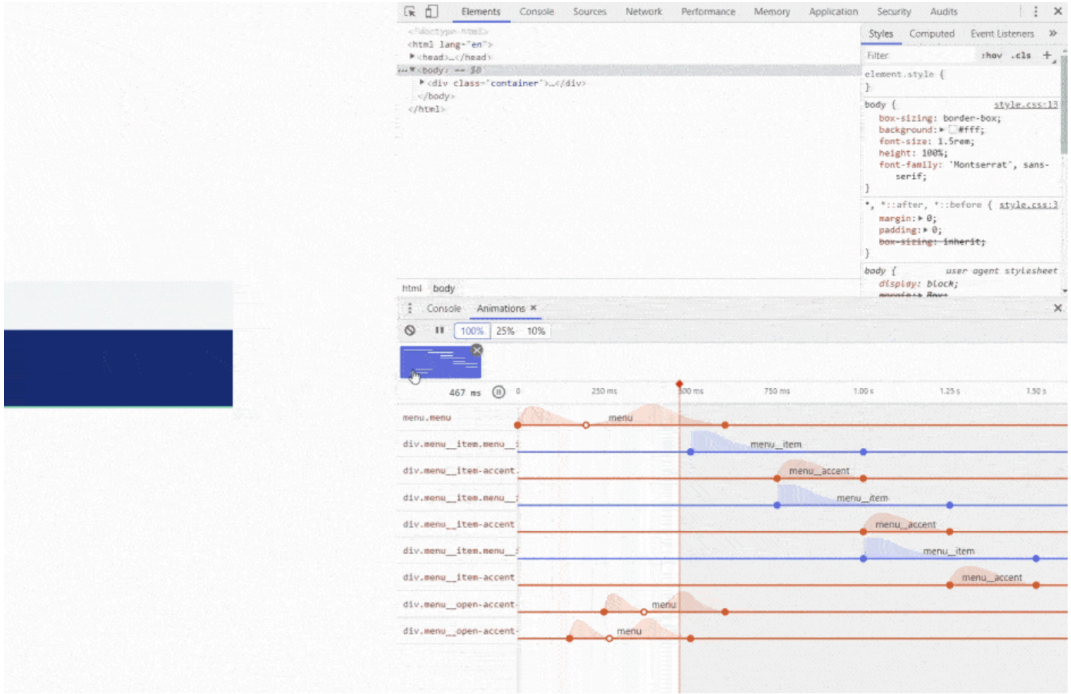
- *-> so, we have an animation which is composed of multiple smaller ones (the entire this made from a series of separate keyframes which are being put together). Those smaller animations are as part of a series which targets the different elements on the page. If you think of an iMovie, each of the elements (the audio / each video section) has its own layer -> we can take the animation, break the elements its operating on down into smaller parts, give each of them its own layer in the stream and then plot the movement of those elements as time goes on and the smaller animations acts on them*
- *-> the 'iMovie' which this section of the course uses to do this for animations which have already been created in an IDE is Chrome DevTools <- this has an animations feature*
- *-> the entire reason of doing this is so that once the a larger animation has been created which consists of multiple stages acting on different elements -> we can break down those elements into different streams and look at their behaviour as the animations act on them*
    - *-> we do this to initialise the start times / durations / delay times of the animations acting on these elements*
    - *->* **THE PREVIOUS PART OF THIS CHAPTER WAS GOING THROUGH THE CODE FOR AN EXAMPLE ANIMATION**
    - **THOUGHT PROCESS FOR USING CHROME DEV TOOLS TO OPTIMISE THE START TIMES / DELAY TIMES / DURATIONS OF THE CONSTITUENT PARTS OF THIS ANIMATION:**
        - ***-> #1 insert the created animation into this tool***
        - ***-> #2 optimise the start times / durations / delay times of the animations acting on these elements by dragging them around as if it's an iMovie we are creating***
        - ***-> #3 export the values for the optimised animations which this produces <- it tells us this in the DOM / console -> as if it's GitHub and it's a change which we want to commit to a branch <- it's telling us where that change is. We export the changes -> the times for the optimised animations***
        - ***-> #4 we enter the times for the optimised animations back into the source code Sass in the IDE -> so then when that webpage is loaded, the times that those smaller animations are set at matches the optimised version which was created using the Chrome animation DevTools***
- ***The structure of the rest of the notes for this chapter:***
    - ***#1 Exploring the UI of the Animation suite in Chrome DevTools to perform this optimisation on the example animation whose code we just explored***
    - ***#2 Notes on the commentary in the course material about performing this optimisation on the example whose Sass / html these notes just explored***
    - ***#3 Notes on the recap section of this chapter***

- ***#1 Exploring the UI of the Animation suite in Chrome DevTools to perform this optimisation***
    - ***commentary***
        - *-> the aim of what we are doing in this chapter section was explored above*
        - *-> this is going through the 'iMovie' like UI in the Animation suite of the Chrome DevTools which is used to perform the optimisation*
        - *-> it is thus less specific to the example animation which was explored in the previous section of the notes on this chapter*
    - ***the UI of the Animation suite in the Chrome DevTools***

- *to open the Animation suite in Chrome DevTools*
  - #1 create the animation you want to optimise in an IDE, o.e e.g CodePen
  - #2 open the webpage with that animation on it using Chrome
  - #3 Cmd-Opt-I Mac, Ctrl-Shift-I Windows
  - #4 Top RHS of page > icon with three dots stacked on top of each other > More tools > animations
  - #5 let it listen for animations
  - #6 play the animations on the page while it is doing this
  - #7 it will generate the time series graphs which look like iMovie graphs



    - -> each time stream on one those graphs corresponds to an element in the animation
    - -> the x(t) curves for those elements
    - -> each time there is a peak on the x(t) curve -> that is when an animation has been applied to it
    - -> you can see there are loads of peaks -> this animation is made is multiple different elements -> and there are multiple animations which act on them
    - -> the shape of the animations which is created by those graphs follows the cubic-beznier curves for the animations which defined them
- *commentary about the UI in those tools which can be used to inspect an animation*
  - *dragging elements to alter their durations and time delays*
    - -> it's like iMovie -> you can drag each of the peaks (corresponding to an animation on an element) so that they happen earlier or later relative to each other. This is altering the time delay for those various animations
    - -> you can also drag the peaks for elements so that they take more time to complete <- altering the duration of that given animation
  - *other features of this UI*
    - -> language which he's using
      - -> the fluidity of the animations
      - -> the console is listening for the animations on the page <- to gather data about the x(t) curves of their elements - in order to produce them
    - *to play the an animation in the UI*
      - -> each animation which loads into the UI will show as it's own box
      - -> in this example it's the blue box
      - -> if you click on that box, then the animation which it reperesents will



play

- -> if you have animations playing on multiple parts of the page, then there will be multiple boxes here (one for each of them)
- -> clicking on one of the boxes there will play the animation which it represents
- -> when this happens, the UI will also scrub through it -> like it is scrubbing through an iMovie (the red line with a dot at the top - playhead - will move from the LHS to the RHS of the page)
- ***the layers each correspond to an element being animated, not to one animation -> a layer per element and a peak per use of an animation on that element***
  - clicking through those different layers highlights its corresponding element on the webpage
  - -> it's playing through the animation on repeat in the 'iMovie'

- *those are the x(t) curves, and each peak corresponds to the use of an animation*
  - when there is no peak and that line is flat, it's called a 'baseline' (which are static most of the time for the elements in this animation)
  - -> you can infer in some cases that an ease-in timing function has been set from the shape of the x(t) curve where the animation has been applied
  - -> the names of the keyframes used to create the animations in this example are shown on the graphs for them in the DevTools
- *there is a panel at the top of the 'iMovie' streams with a percentage in it, this is for the percentage speed of the animations in the timeline as a function of how quickly they normally play*
- you can slow the speed of an animation by reducing its duration

- ***further commentary***
  - -> this can't be used to change the percentages in keyframes
  - **-> the animation delay times which were updated are now in the elements pane**
  - -> diffing
    - -> change one of the animations which has been inspected in the animations section of the Chrome DevTools
      - those DevTools can't change the keyframe percentages for the animations -> just those times
      - -> you are changing the durations / delays of those animations, by

clicking / dragging the animated element in the pane
    - -> in the Elements section in those DevTools <- you can see the modified values of those durations which would be entered back into the Sass from the console
        - -> so you change the duration of the animated elements in the console by dragging the different peaks for those elements in the 'iMovie' infertace in the DevTools
        - -> that outputs values for the duration and delay times
        - -> those values can be put into the Sass
    - this section of the panel (the Elements tab) can also be used to tell you which exact changes need to be made to the source code
    - -> the elements tab contains a link to the code in the DOM
        - -> then local modifications from the context menu <- this shows the changes which were made from the code in that section, which it highlights
        - the changes panel opens next the the animations panel
        - this is similar to merge conflicts
        - original values are highlighted in red, and updated values for the changed property are highlighted in green
- -> *when using this animation for the*
- ***#2 Notes on the commentary in the course material about performing this optimisation on the example whose Sass / html these notes just explored***
    - -> the Sass / css for an example was first explored
    - -> then the UI for inspecting this in the Chrome DevTools Animations pane
    - *-> the approach which is used with this is*
        - -> the animation which we are inspecting is made from a combination of animations (in this case) acting on smaller elements
        - -> the animations which are acting on those smaller elements have start times and durations
        - -> we aren't playing around with how those animations are being defined (the keyframes), we're playing around with the delay and duration each time one of them is used
        - -> the Chrome DevTools in this case are inspecting an animation which has already been created
        - -> we are playing around with each of the animations in that pane, in order to optimise their durations and start times
    - *-> this is a description of the process which is used to fine-tune those values*
        - -> the UI for the Animations panel in the Chrome DevTools is like an iMovie
        - -> see #1 for using scrubbing through the x(t) curves for each of these animations in this UI
        - *-> you are going through a process, which is*
            - -> change one of the instances where an animation has been applied
                - -> either in terms of its duration or delay
            - -> look at how the animation behaves with those parameters
            - -> repeat this until happy with the result
        - *-> things to pay attention to when looking at the animation with newly tweaked duration / delay times*
            - -> if certain items are coming in too late / early
            - -> if we want animations on elements to be evenly staggered or if we want their delays to be offset
            - -> if some elements are moving too quickly -> and we need to increase the durations of those animations so the elements they're acting on move slower,

e.g
    - -> going through a tweaking process <- to initialise those duration / delay values
- -> *further commentary about after having initialised the animation duration and delay times using this approach*
    - -> initialising the animation is a process
        - you can choose the delay and duration times which you think it should have based off of experience, but then these require tweaking
        - -> he's listed out the changes he made to the animation for this example in the course
            - -> the changes from that process can be seen in the Animations tab in Chrome DevTools <- they are shown here like changes to a GitHub repo
            - -> he then gives the code which was explored in the beginning of these notes
    - -> once an animation is created -> getting the user's feedback on it
- ***#3 Notes on the recap section of this chapter***
    - -> refinement / riffing for Sass animations
        - -> initialising the duration of animations from values based off of experience and the delay times for them
        - -> and then inspecting them in the Animations panel in the DevTools in Chrome and fine tuning them there
    - -> the entire chapter is about the Sass Animation pane in Chrome DevTools
        - -> for the duration / delay of animations
        - -> the Changes panel lets you see the properties that you've changed and what their updated values are -> so you can use them to change the source code they come from
        - -> the values of the animation that were being changed by riffing through them in the Animation pane in the Chrome DevTools were their durations and delays <- the UI of this Animation pane looks like an iMovie edit on the animation which is being inspected