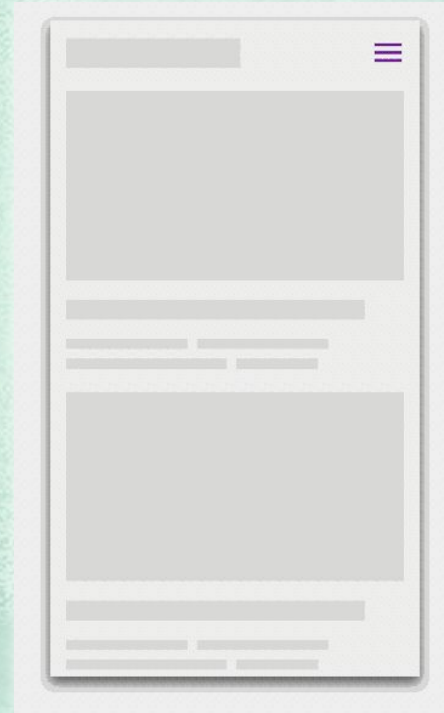
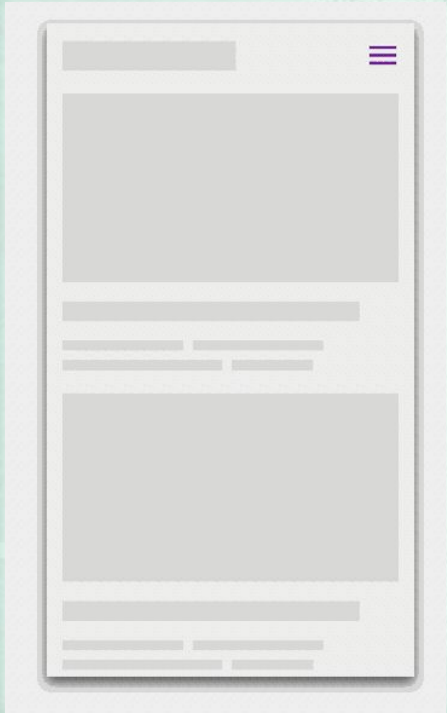


PaSe: An Extensible and Inspectable DSL for Micro-Animations

Ruben Pieters & *Tom Schrijvers*

What are Micro-Animations?



<https://hike.one/update/why-use-micro-animations-in-your-design>

They can get very complex...

DICEY DUNGEONS

INNATE: Cornelius gets an extra dice every turn.

Cornelius x0
80/80

LIQUORICE+
Add 3 poison

BOOP
Do 3 damage

SWEETS
Heal 2 health

NEXT UP (8 left)
Sweets
Robot
Buzzer
SNAP!
Exchange matching cards for dice

LIMIT

Jester x3
34/34

END TURN

Ok, let's get started!

7 Best Animation Libraries for UI Designers 2018 | Codementor

<https://www.codementor.io/hayeskier/7-best-animation-libraries-for-ui-designers-2018...>

Jun 29, 2018 · Overall, a great **animation library** for UI designers in 2018. A must-try. AnimeJS. GitHub. AnimeJS is fairly new JS **animation library** compared to others listed here. In the small span of its release, it has grown rapidly and also shows promise to become of the **best** out there. AnimeJS is a complete package when it comes to **animation library**.

The 10 best JavaScript libraries for SVG animation

<https://noeticforce.com/javascript-libraries-for-svg-animation>

Sep 15, 2019 · The 10 **Best JavaScript Libraries for SVG Animation** VelocityJS. Walkway. RaphaelJS. Snap.Svg. Bonsai. Lazy Line Painter. Vivus. Progress.js. Conclusion.

Top 20 jQuery Animation Library and Plugins 2019 - Colorlib

<https://colorlib.com/wp/jquery-animation-library-plugins/>

Mar 28, 2019 · Top 20 **jQuery Animation Library** and Plugins 2019 Icon Animation. Powered by mo.js. Motion Graphics for the Web with mo.js. Polaroid Stack to **Animation**. Material Scroll **Animation**. Elastic Circle Slideshow. Interactive Bar **Animation**. pageSwitch for JavaScript. Animating an SVG Menu Icon ...

15 Best JavaScript Animation Libraries for Developers | Code ...

<https://codegeekz.com/15-best-javascript-animation-libraries-for-developers/>

For this round-up, we would like to introduce some of these innovative uses and benefits of jQuery when it comes to **animation** effects for your projects, websites and apps. Enjoy!
1. Tween JS. TweenJS is a simple tweening **library** for use in Javascript. It was developed to integrate well with the EaselJS **library**, but is not dependent on or ...

9 of the Best Animation Libraries for UI Designers — SitePoint

<https://www.sitepoint.com/our-top-9-animation-libraries/>

Animation is a part of a UI designer's job. Here are 9 free **animation** libraries we think you should know about. They offer the most power for the smallest file size, while being relatively easy to use for ...

10 Best Free Animation Libraries For The Web | Webdesigner Depot

<https://www.webdesignerdepot.com/2018/01/10-best-free-animation-libraries-for-the-...>

Some of the key trends for 2018, but you don't have to reinvent the wheel for your sites. We look at the best free **animation** libraries for your sites. You can do some crazy things with UI **animations** on the web. This is a list of the best modern websites ...

jQuery Animation Library - Your Partner for Quality Animations

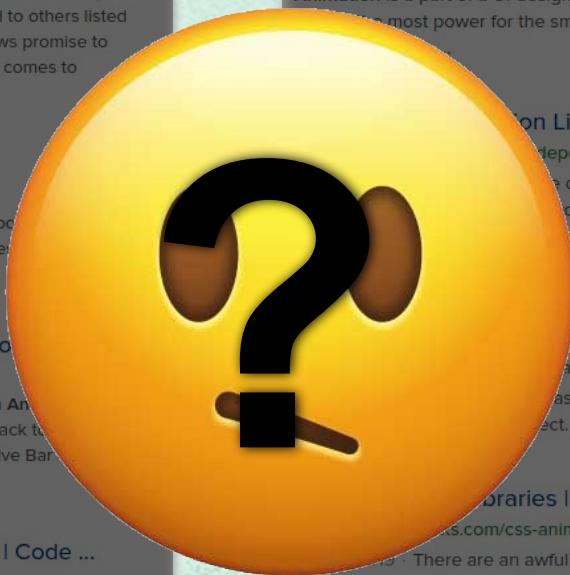
jquery.com

jQuery Animation Library is easy. We offer a wide variety of pre-rigged Characters and Animations to help you get started. Discover The **Animation Library** today.

10 Best CSS Animation Libraries | CSS-Tricks

<https://css-tricks.com/css-animation-libraries/>

There are an awful lot of libraries that want to help you animate things on the web. These aren't really libraries that help you with the syntax or the technology of **animations**, but rather are grab-and-use as-is libraries.



Structure in a DSL


- Atomic Operations
- Combination Facilities

Operations


anime 3.1.0 SVG

TARGETS
PROPERTIES
PROPERTY PARAMETERS
ANIMATION PARAMETERS
VALUES
KEYFRAMES
STAGGERING
TIMELINE
CONTROLS
CALLBACKS & PROMISES
SVG
| MOTION PATH
| MORPHING
| LINE DRAWING
EASINGS
HELPERS


MOTION PATH



MORPHING




LINE DRAWING



EASINGS

LINEAR



MOTION PATH

Animates an element relative to the x, y and angle values of an SVG path element.

```
var myPath = anime.path('svg path');
```

The path function returns a new Function that returns the specified property.

Motion path animations are responsive since v3

PARAMETERS	EXAMPLE	INFO
'x'	myPath('x')	Return the current x value in 'px' of the SVG path
'y'	myPath('y')	Return the current y value in 'px' of the SVG path
'angle'	myPath('angle')	Return the current angle value in 'degrees' of the SVG path

CODE EXAMPLE

```
var path = anime.path('.motion-path-demo path');  
  
anime({  
  targets: '.motion-path-demo .el',  
  translateX: path('x'),  
  translateY: path('y'),  
  rotate: path('angle'),  
  easing: 'linear',  
  duration: 2000,  
  loop: true  
});
```

Download Star 33,193

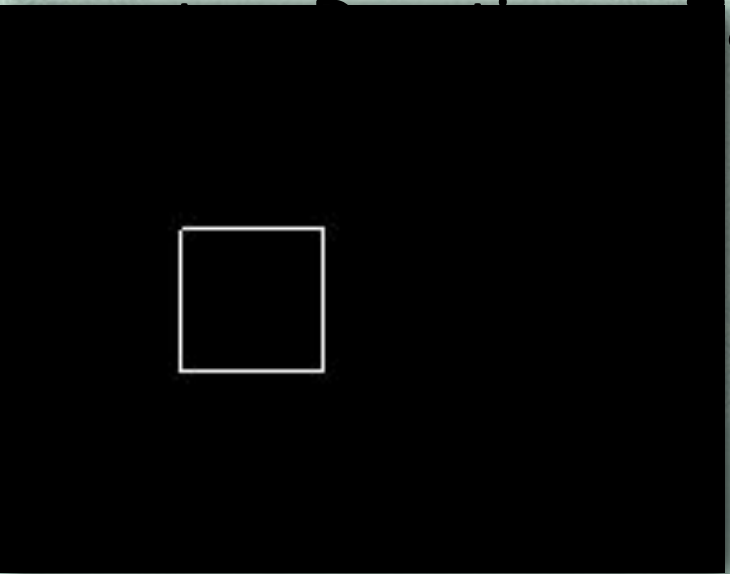
<https://animejs.com/documentation>

Operations

```
r = linearTo (box . x) (For 0.5) (To 50)
```

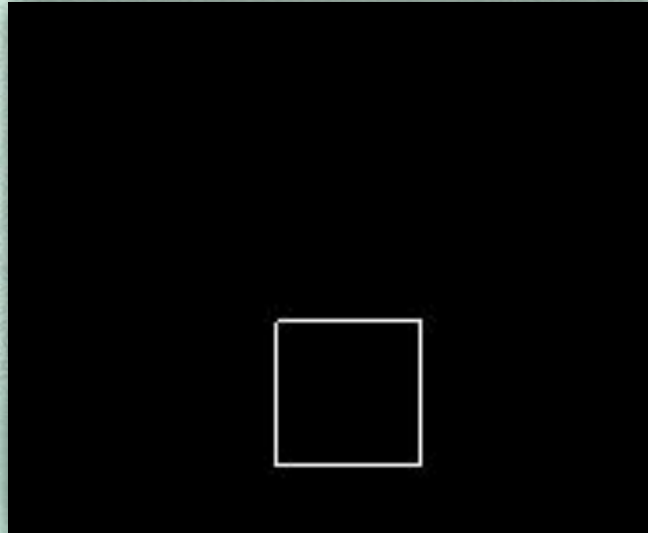
Diagram illustrating the components of the `linearTo` function call:

- `(box . x)` is labeled as `P` (Position).
- `(For 0.5)` is labeled as `Distance`.
- `(To 50)` is labeled as `target`.



Operations

```
u = linearTo (box . y) (For 0.5) (To 50)
```



Structure in a DSL

- Atomic Operations
- **Combination Facilities**

Combination

anime 3.1.0

CALLBACKS & PROMISES

BEGIN & COMPLETE

begin() callback is triggered once, when the animation starts playing.

complete() callback is triggered once, when the animation is completed.

Both begin() and complete() callbacks are called if the animation's duration is 0.

TYPE	PARAMETERS	INFO
Function	animation	Return the current animation Object

CODE EXAMPLE

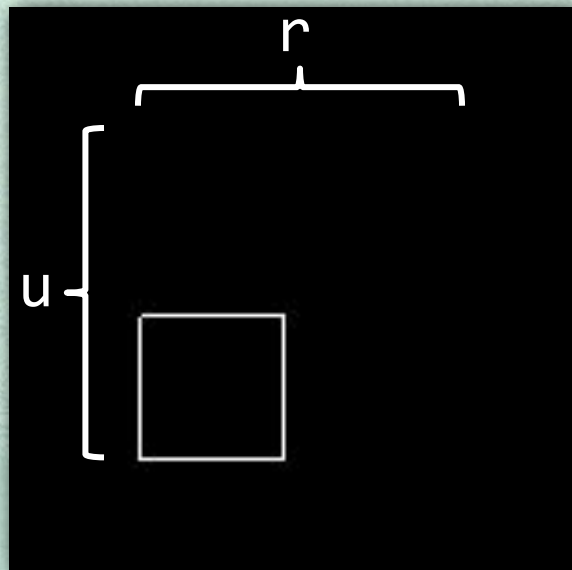
```
anime({
  targets: '.begin-complete-demo .el',
  translateX: 240,
  delay: 1000,
  easing: 'easeInOutCirc',
  update: function(anim) {
    progressLogEl.value = 'progress : ' + Math.round(anim.progress) + '%';
    beginLogEl.value = 'began : ' + anim.began;
    completeLogEl.value = 'completed : ' + anim.completed;
  },
  begin: function(anim) {
    beginLogEl.value = 'began : ' + anim.began;
  },
  complete: function(anim) {
    completeLogEl.value = 'completed : ' + anim.completed;
  }
});
```

Download Star 33,235

<https://animejs.com/documentation>

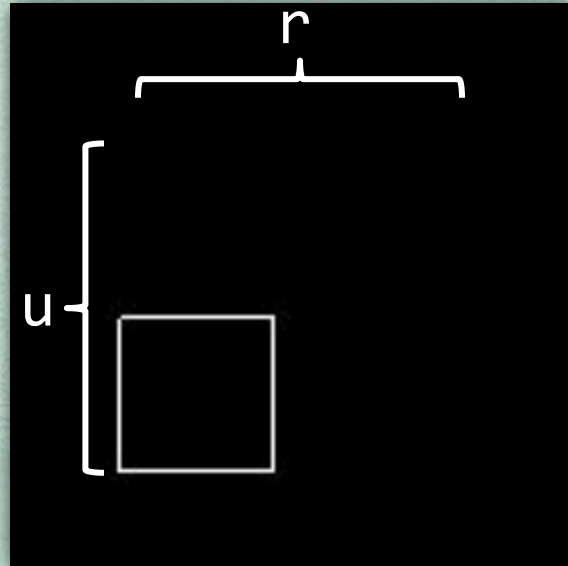
Combination

`upThenRight = u `sequential` r`



Combination

diagonal = u `parallel` r

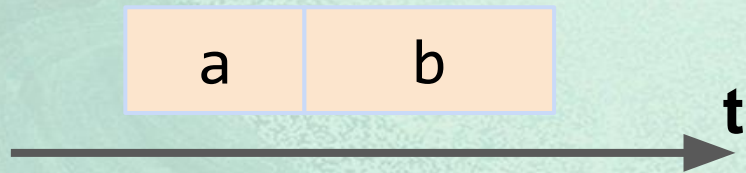


Combinator
Callback **FIGHT** style
Style

Round 1: Semantics

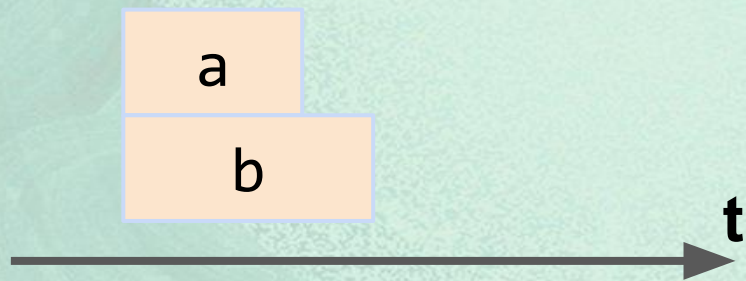
Semantics

a `sequential` b



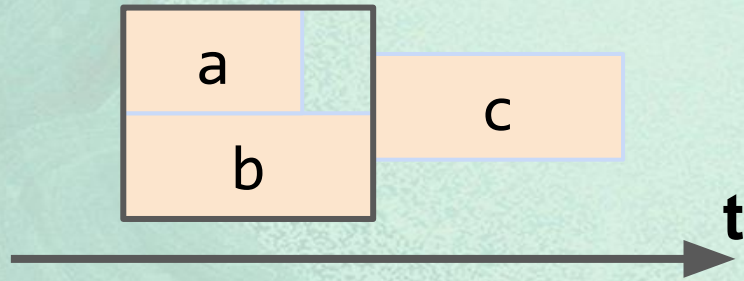
Semantics

a `parallel` b



Semantics

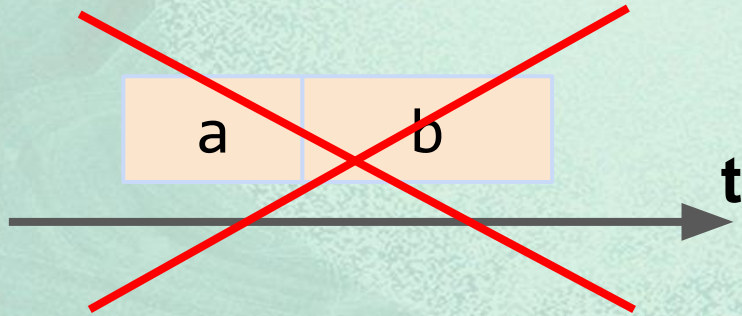
(a `parallel` b) `sequential` c



Semantics

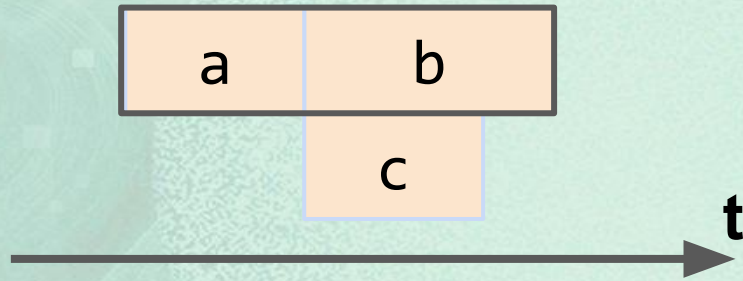
`a`onComplete` b`

`(= a.onComplete(() => b.play()))`

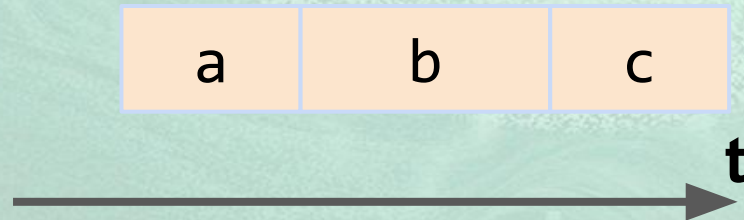


Semantics

(a `onComplete` b) `onComplete` c

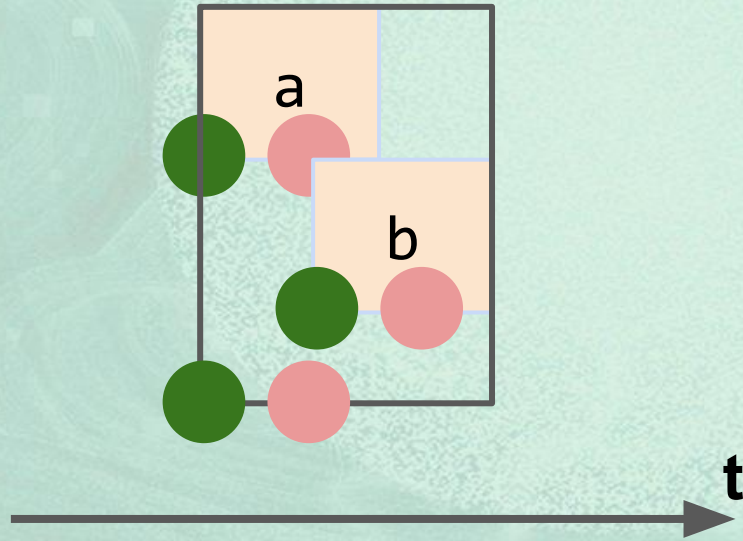


a `onComplete` (b `onComplete` c)



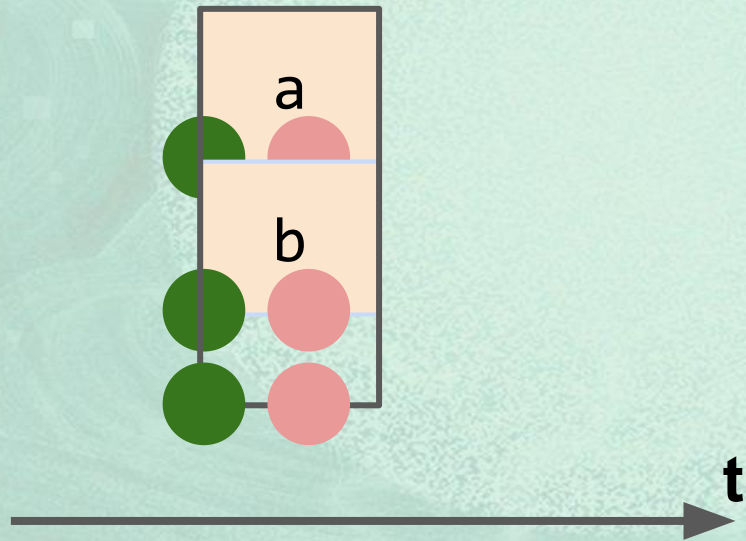
Semantics

a `onComplete` b



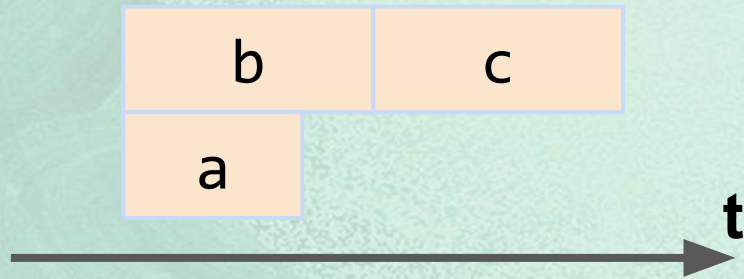
Semantics

a `onBegin` b



Semantics

`(b onComplete c) `onBegin` a`



Parallel/Sequential Combinators

The screenshot shows the Ren'Py Documentation website. At the top left is the Ren'Py logo and the text "Ren'Py Documentation". To its right are links for "Home Page" and "Online Documentation". A search bar is located at the top right. On the left side, there is a navigation menu with categories like "Animation and Transformation Language", "Ren'Py Script Statements", "ATL Syntax and Semantics", and "ATL Statements". The main content area is titled "Parallel Statement" and contains the following text:

The `parallel` statement is used to define a set of ATL blocks to execute in parallel.

```
atl_parallel ::= "parallel" ":"  
              atl_block
```

Parallel statements are greedily grouped into a parallel set when more than one parallel statement appears consecutively in a block. The blocks of all parallel statements are then executed simultaneously. The parallel statement terminates when the last block terminates.

The blocks within a set should be independent of each other, and manipulate different properties. When two blocks change the same property, the result is undefined.

```
show logo base:  
  parallel:  
    xalign 0.0  
    linear 1.3 xalign 1.0  
    linear 1.3 xalign 0.0  
    repeat  
  parallel:  
    yalign 0.0  
    linear 1.6 yalign 1.0  
    linear 1.6 yalign 0.0  
    repeat
```

Below the main content, there are two sidebars. The left sidebar has a "Qt" header and a list of links: "All QML Types", "All Qt Modules", "Qt Creator Manual", and "All Qt Reference Documentation". The right sidebar has a "Getting Started" header and a list of links: "Getting Started with Qt", "What's New in Qt 5", "Examples and Tutorials", "Supported Platforms", and "Qt Licensing".

At the bottom right of the page, there are links for "Blog", "Contact Us", and a user profile icon.

The following animation runs two number animations in parallel. The `Rectangle` moves to (50,50) by animating its `x` and `y` properties at the same time.

```
import QtQuick 2.0  
  
Rectangle {  
  id: rect  
  width: 100; height: 100  
  color: "red"  
  
  ParallelAnimation {  
    running: true  
    NumberAnimation { target: rect; property: "x"; to: 50; duration: 1000 }  
    NumberAnimation { target: rect; property: "y"; to: 50; duration: 1000 }  
  }  
}
```

Timeline



Sequencing with Timelines

Choreographing complex sequences is crazy simple with GSAP's [Timelines](#).

A timeline is a **container** for [tweens](#) where you place them in time (like a schedule). They can overlap or have gaps between them; you have total control. As the timeline's playhead moves, it scrubs across its child tweens and renders them accordingly! Insert as many as you want and control the entire group as a whole with the standard methods (`play()` , `reverse()` , `pause()` , etc.). You can even nest timelines within timelines!

Once you get the hang of timelines, a whole new world of possibilities will open up. They provide a fantastic way to [modularize your animation code](#).

When to Use a Timeline

- To control a group of animations as a whole.
- To build a sequence without messing with lots of `delay` values (progressively build so that timing adjustments to earlier animations automatically affect later ones, greatly simplifying experimentation and maintenance).
- To [modularize your animation code](#).
- To do any kind of complex choreographing.
- To fire callbacks based on a group of animations (like "after all of these animations are done, call `myFunction()` ").

<https://greensock.com/get-started>

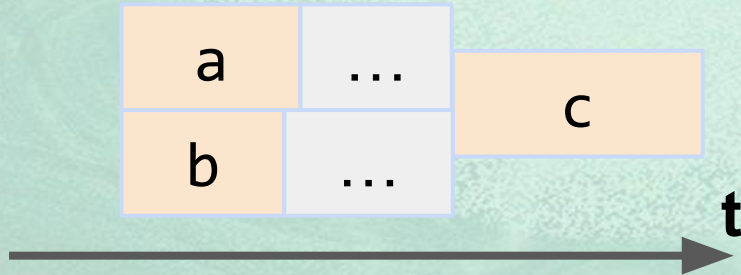
Round 2: Expressivity

Expressivity

(b `onComplete` c) `onBegin` a

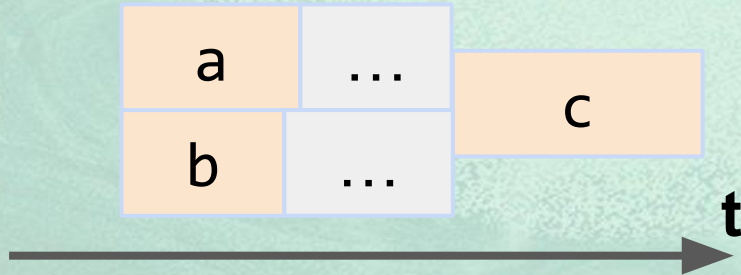
OR

(a `onComplete` c) `onBegin` b



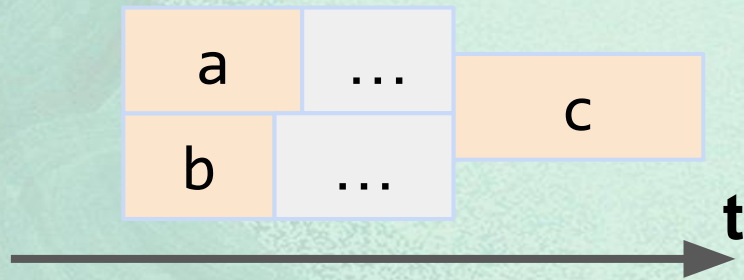
Expressivity

```
if duration a > duration b  
  then (a `onComplete` c) `onBegin` b  
  else (b `onComplete` c) `onBegin` a
```



Expressivity

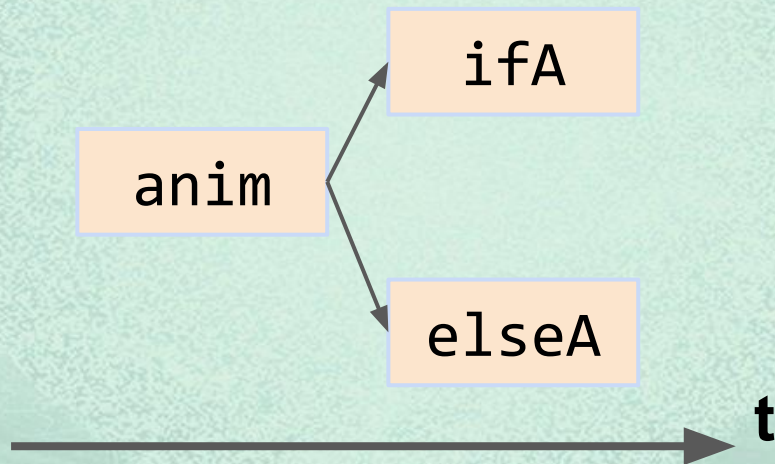
(a `parallel` b) `sequential` c



Expressivity

```
function ifThenElse(anim, cond, ifA, elseA){  
  anim.onComplete(() => if (cond()) {  
    ifA.play()  
  } else {  
    elseA.play()  
  });  
  return anim;  
}
```

Expressivity



Expressivity

`ifThenElse anim cond ifA elseA = ?`

Expressivity

```
a `onComplete` b
```

```
onComplete ::
```

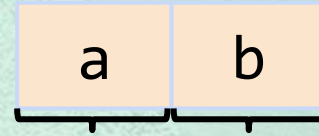
```
  Animation -> (() -> IO ()) -> Animation
```

```
play :: Animation -> IO ()
```

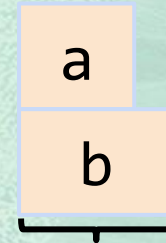

Round 3: Inspectability

Inspectability

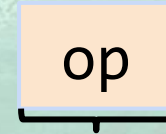
`duration (a `sequential` b) =
duration a + duration b`



`duration (a `parallel` b) =
max (duration a) (duration b)`



`duration op = <basic value>`



Inspectability

```
function duration(anim) {  
  const basicDur = <basic value>;  
  // analyze all callbacks to fetch  
  // additional duration  
  const additionalDur = ...;  
  return basicDur + additionalDur;  
}
```

Inspectability



GSAP 3

```
anim1
```

```
.add(() => anim2.play())  
.add(() => anim3.play());
```

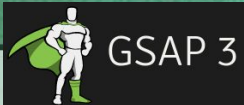
anim1

anim2

anim3



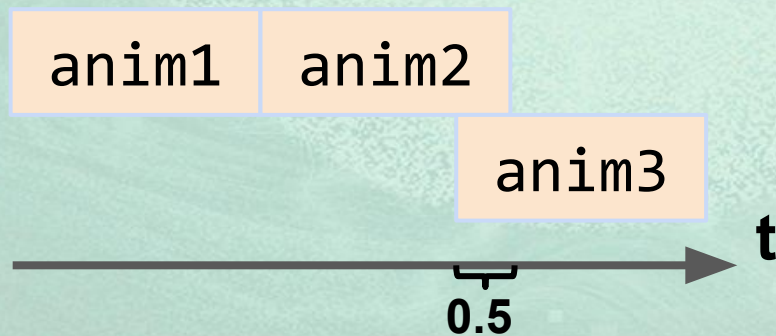
Inspectability



anim1

```
.add(() => anim2.play())
```

```
.add(() => anim3.play(), "-=0.5");
```

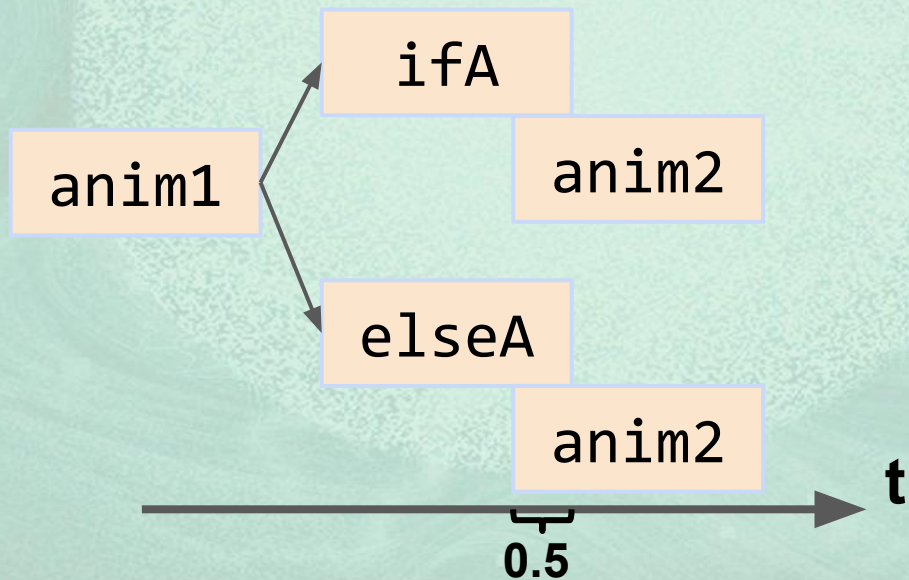


Inspectability



```
ifThenElse(anim1, cond, ifA, elseA)  
  .add(() => anim2.play(), "-=0.5");
```

Expected:

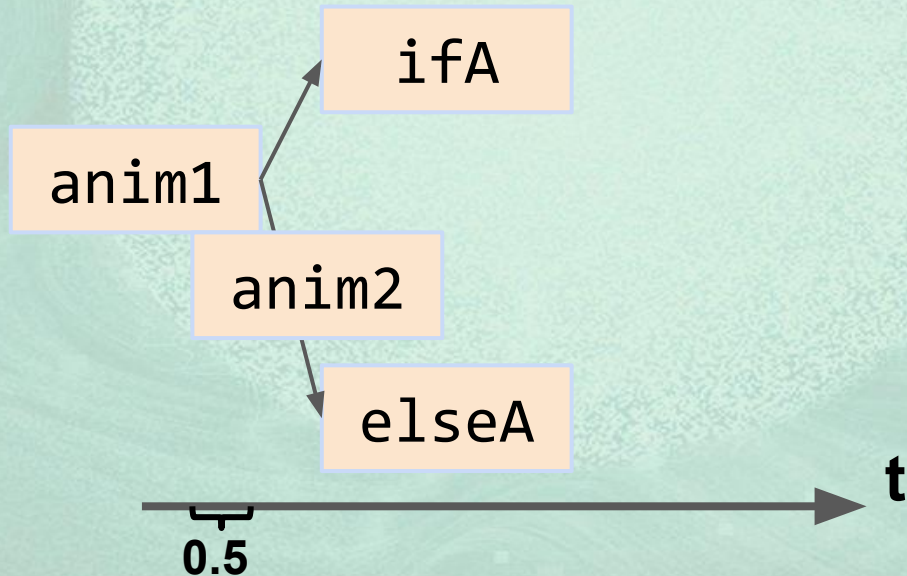


Inspectability



```
ifThenElse(anim1, cond, ifA, elseA)  
  .add(() => anim2.play(), "-=0.5");
```

Reality:



Inspectability



```
ifThenElse(anim1, cond, ifA, elseA)  
  .add(() => anim2.play(), "-=0.5");
```

Reality:

anim1

duration = 0

t

A diagram illustrating the execution of an animation. A horizontal arrow points to the right, labeled 't' at its tip. A large black rectangle is positioned above the arrow, representing an animation. The text 'anim1' is written in a light orange box to the left of the rectangle. Inside the black rectangle, the text 'duration = 0' is written in white. The arrow starts at the left edge of the rectangle and ends at its right edge, indicating that the animation's duration is zero.

Solution

Open Description

`u `sequential` r`

`u `parallel` r`



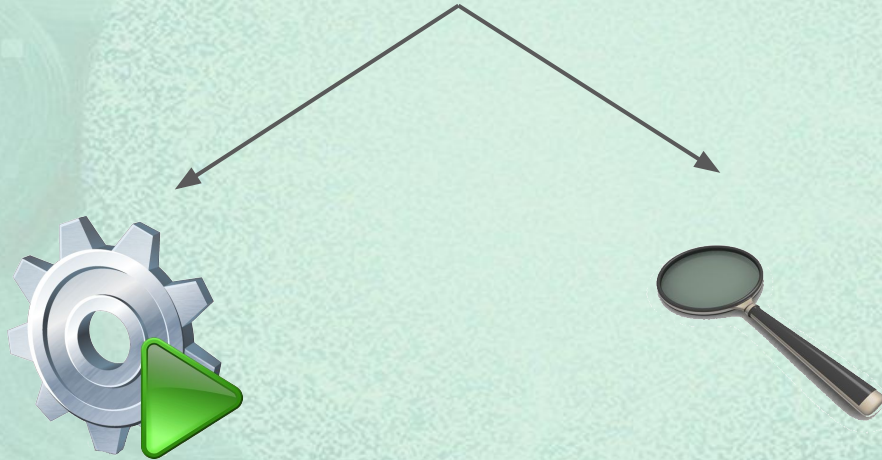
`ifThenElse anim cond ifA elseA`



`...`

Interpretation

ifThenElse cond ifA elseA



Interpretation - Animate

```
ifThenElse cond ifA elseA
```



```
do
```

```
  bool <- cond
```

```
  if bool then ifA else elseA
```


Interpretation - Inspection

`ifThenElse cond ifA elseA`



`duration (ifThenElse cond ifA elseA)`
`= ?`

Interpretation - Inspection

`ifThenElse cond ifA elseA`



```
duration (ifThenElse cond ifA elseA)
= if (duration ifA == duration elseA)
  then duration ifA
  else error
```


Interpretation - Inspection

`ifThenElse cond ifA elseA`


$$\begin{aligned} \text{maxDur } (\text{ifThenElse cond ifA elseA}) \\ = \text{max } (\text{maxDur ifA}) (\text{maxDur elseA}) \end{aligned}$$

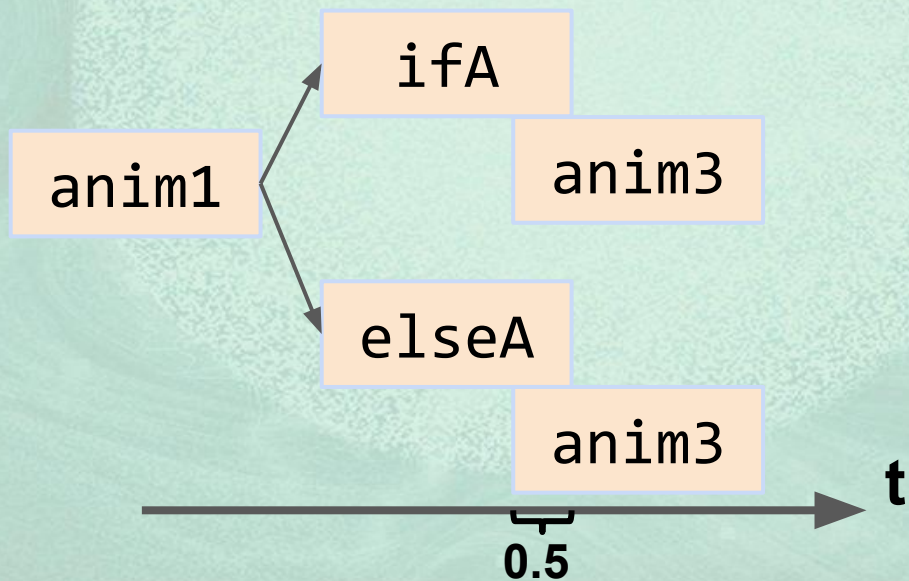
Relative Sequential Composition

relSequential

```
(anim1 `sequential` (ifThenElse cond ifA elseA))
```

anim3

-0.5



Conclusion

- Keep your semantics in mind to avoid unintuitive behaviour
- Enable inspectability with structured representation
- Keep expressivity by using an open encoding



rubenpieters



ruben.pieters@cs.kuleuven.be

<https://dtai.cs.kuleuven.be/events/fpcourse>
fpcourse@cs.kuleuven.be

Functional Programming & Domain-Specific Languages