CSS Animation

Visual Change Over Time

R. Scott Granneman

Jans Carton

Notes & URLs for this presentation can be found…

- underneath the link to this slide on granneman.com
- at files.granneman.com/presentations/webdev/CSS-Animation.txt

Data Types

4 data types specifically useful with animation

<shape>
<time>
<timing-function>
<angle>

<shape>

`<shape>`

Represents a *rectangular region* to which the `clip` property is applied

Defined using the `rect()` function

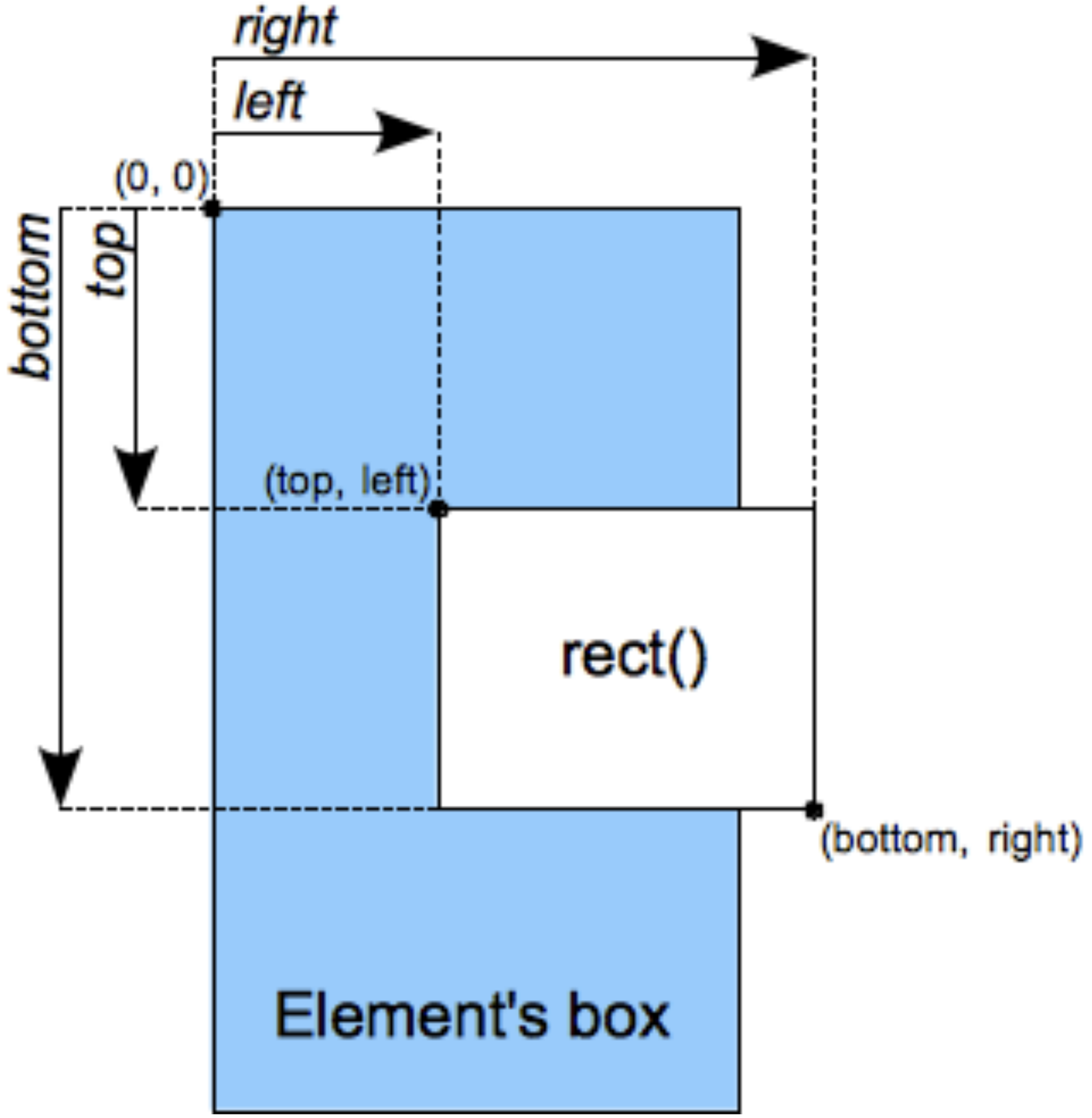
`rect(top, right, bottom, left)`

`top`: `<length>` of distance between *top* of rectangle & *top* border of its container box

`right`: `<length>` of distance between *right* of the rectangle & *left* border of its container box

`bottom`: `<length>` of distance between *bottom* of rectangle & *top* border of its container box

`left`: `<length>` of distance between *left* of the rectangle & left *border* of its container box


right
left
(0, 0)
bottom
top
(top, left)
rect()
(bottom, right)
Element's box

						
`<shape>`	8	1.3	1	1	?	?

<time>

`<time>`

Represents *time*, which keeps on slipping into the future: a `<number>` immediately followed by a unit

Units
» `s`: second
» `ms`: millisecond (`1000ms` = `1s`)

Valid	Invalid
`7s`	`0`
`-7ms`	`7`
`7.7s`	`7 s`
`+0s`	

`0` needs a unit

Needs a unit

No spaces

						
`<time>`	9	<3.2	<11	4	?	?

<timing-function>

`<timing-function>`

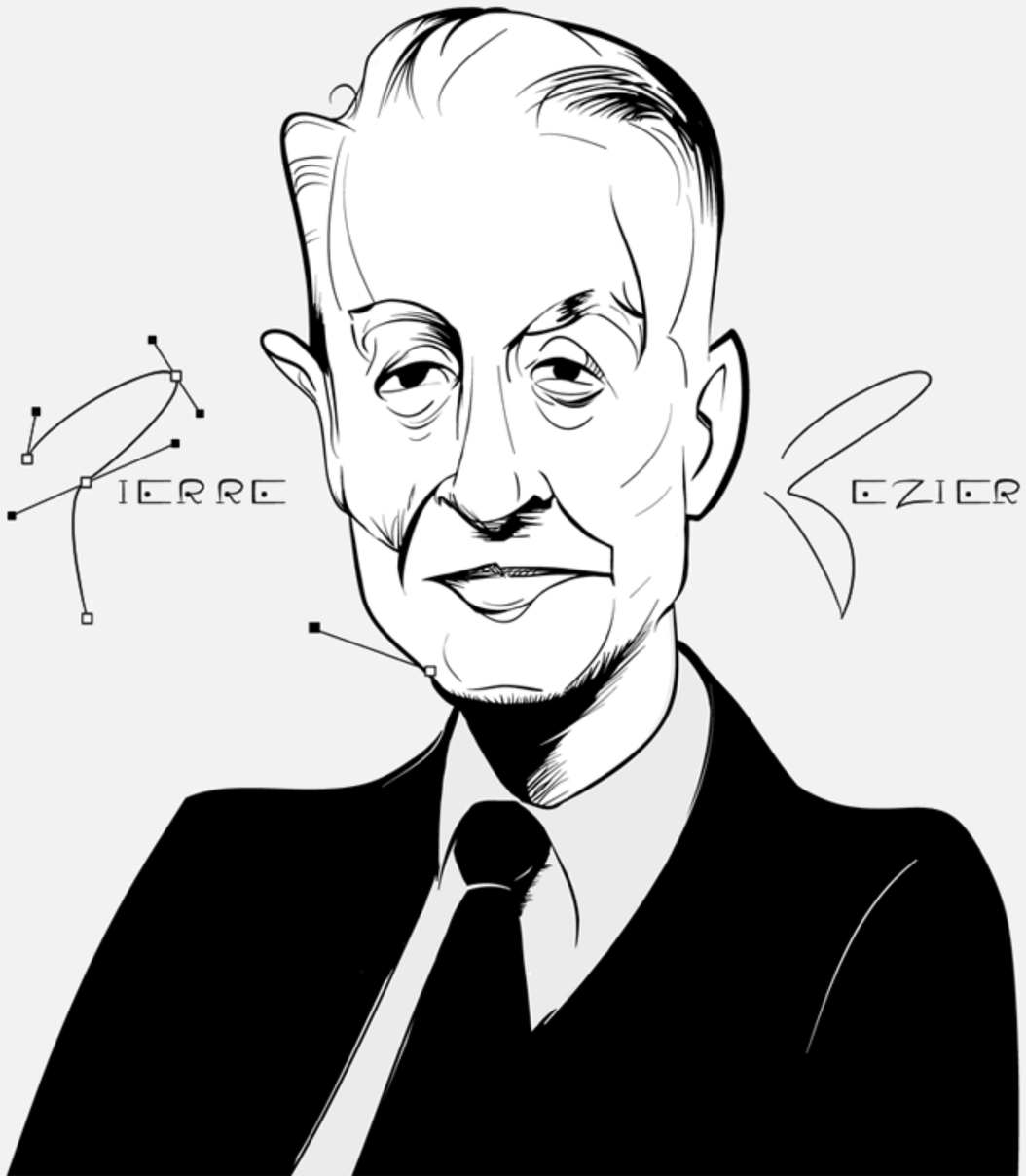
Represents an acceleration curve showing *speed of change over time* during animations & transitions

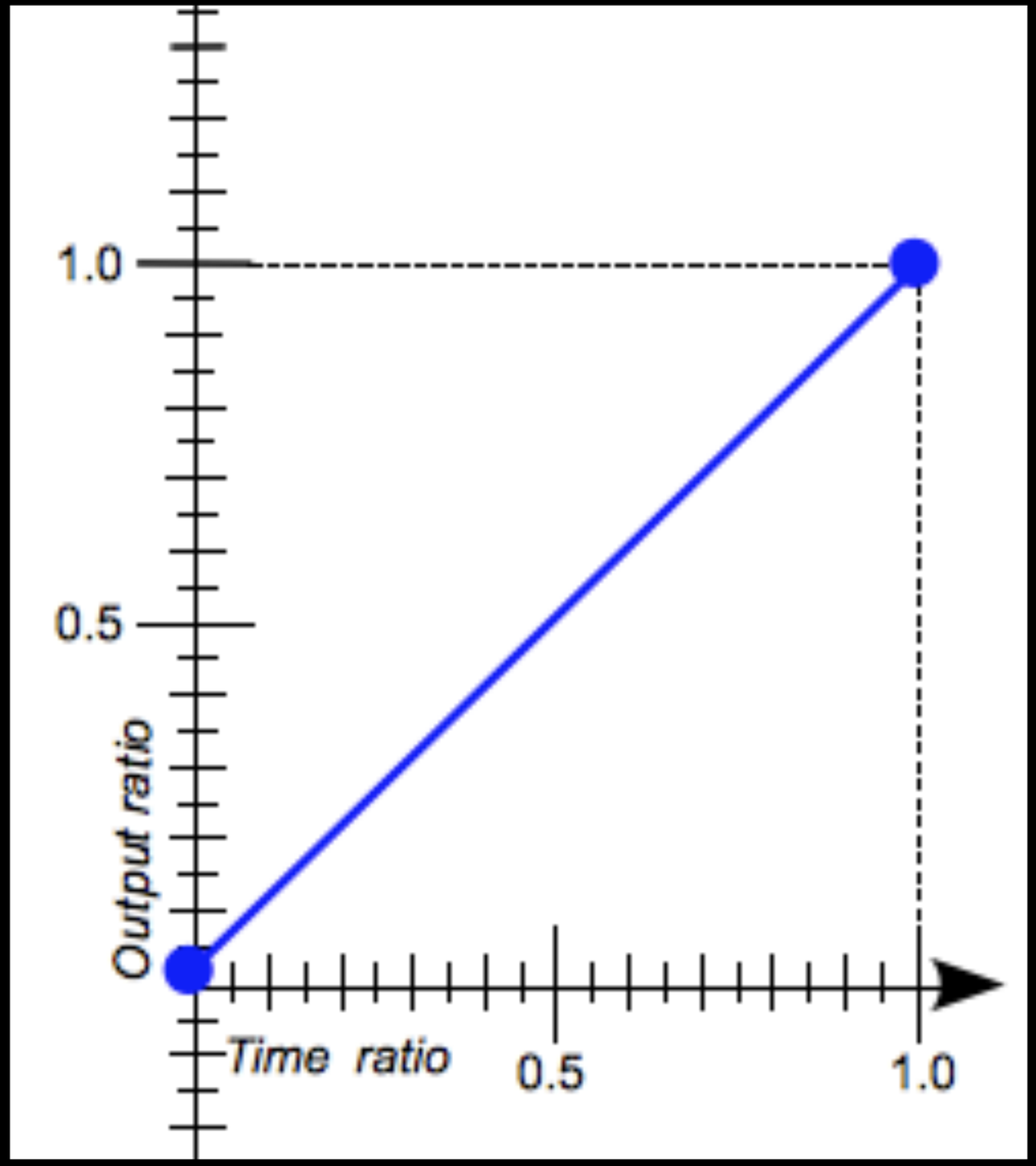
2 kinds of timing functions:

» *cubic Bézier curves* (AKA *easing functions*)
» *staircase functions*: equidistant steps

Values for cubic Bézier curves

- `linear`
- `ease`
- `ease-in`
- `ease-out`
- `ease-in-out`
- `cubic-bezier(x1, y1, x2, y2)`

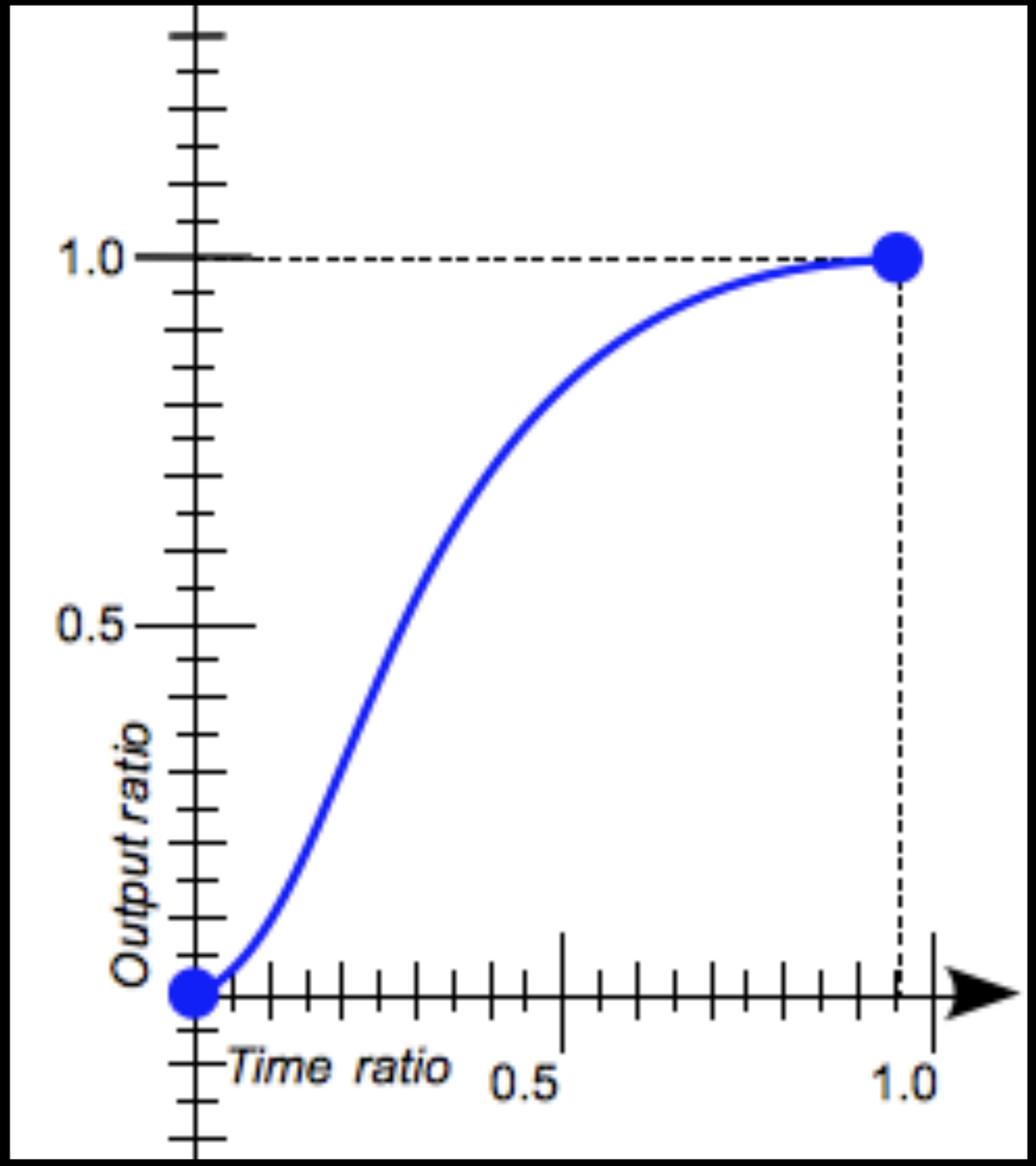

PIERRE
BEZIER



`linear`

Constant speed

Equivalent to `cubic-bezier(0.0, 0.0, 1.0, 1.0)`



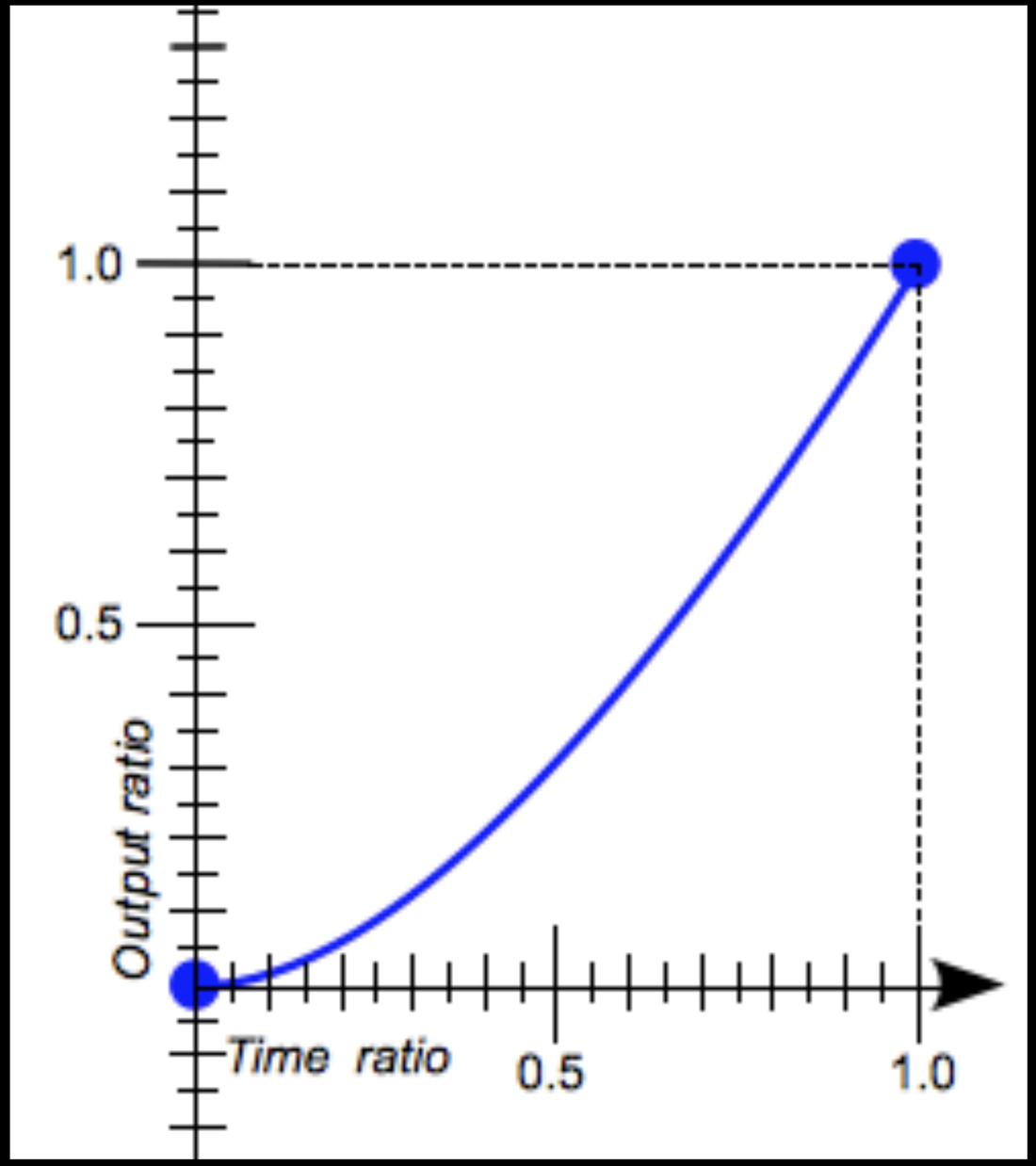
`ease`

Accelerates at beginning

Starts to slow by middle

Equivalent to:

`cubic-bezier(0.25, 0.1, 0.25, 1.0)`



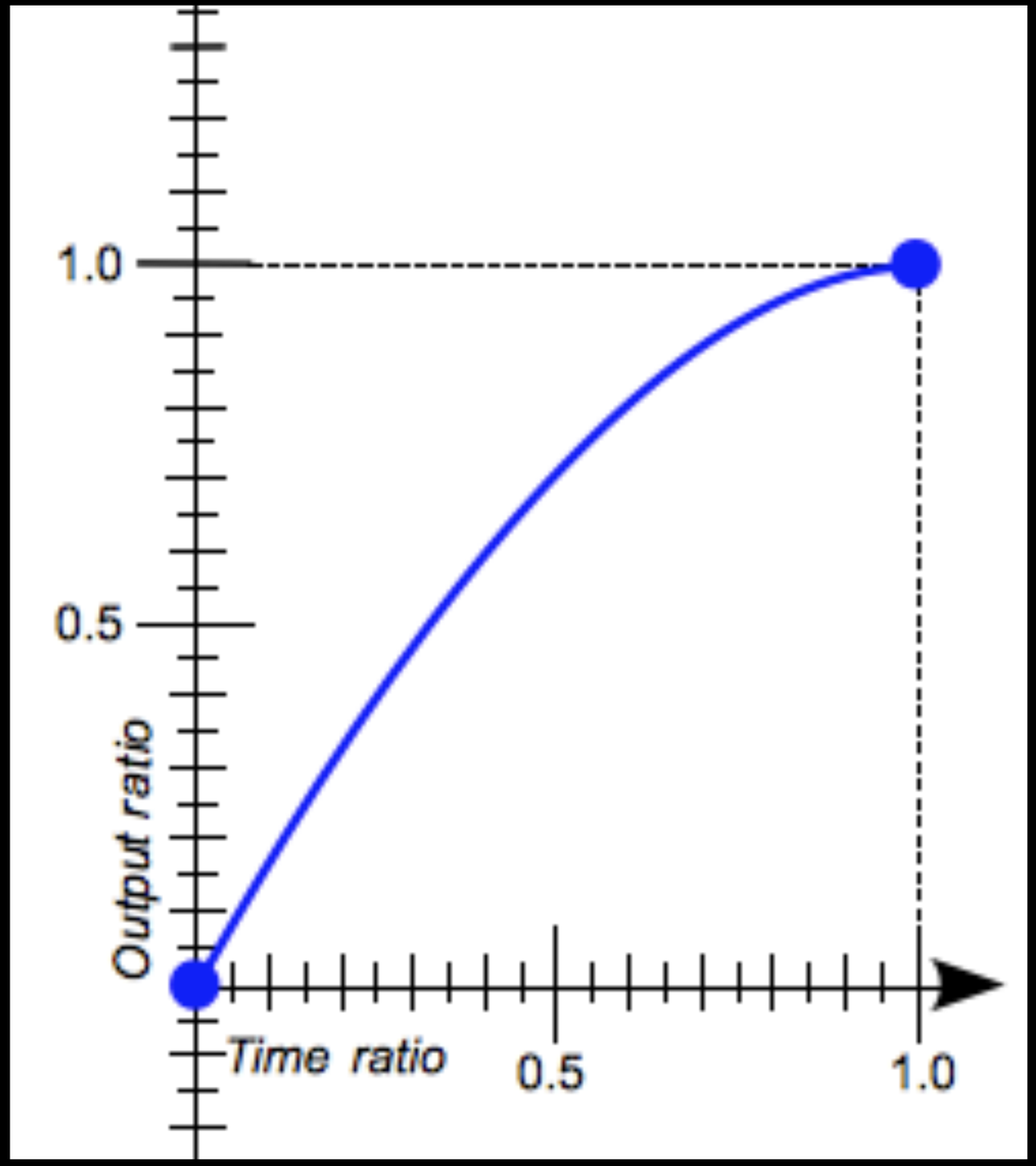
`ease-in`

Begins slowly

Accelerates progressively

Stops abruptly

Equivalent to `cubic-bezier(0.42, 0.0, 1.0, 1.0)`

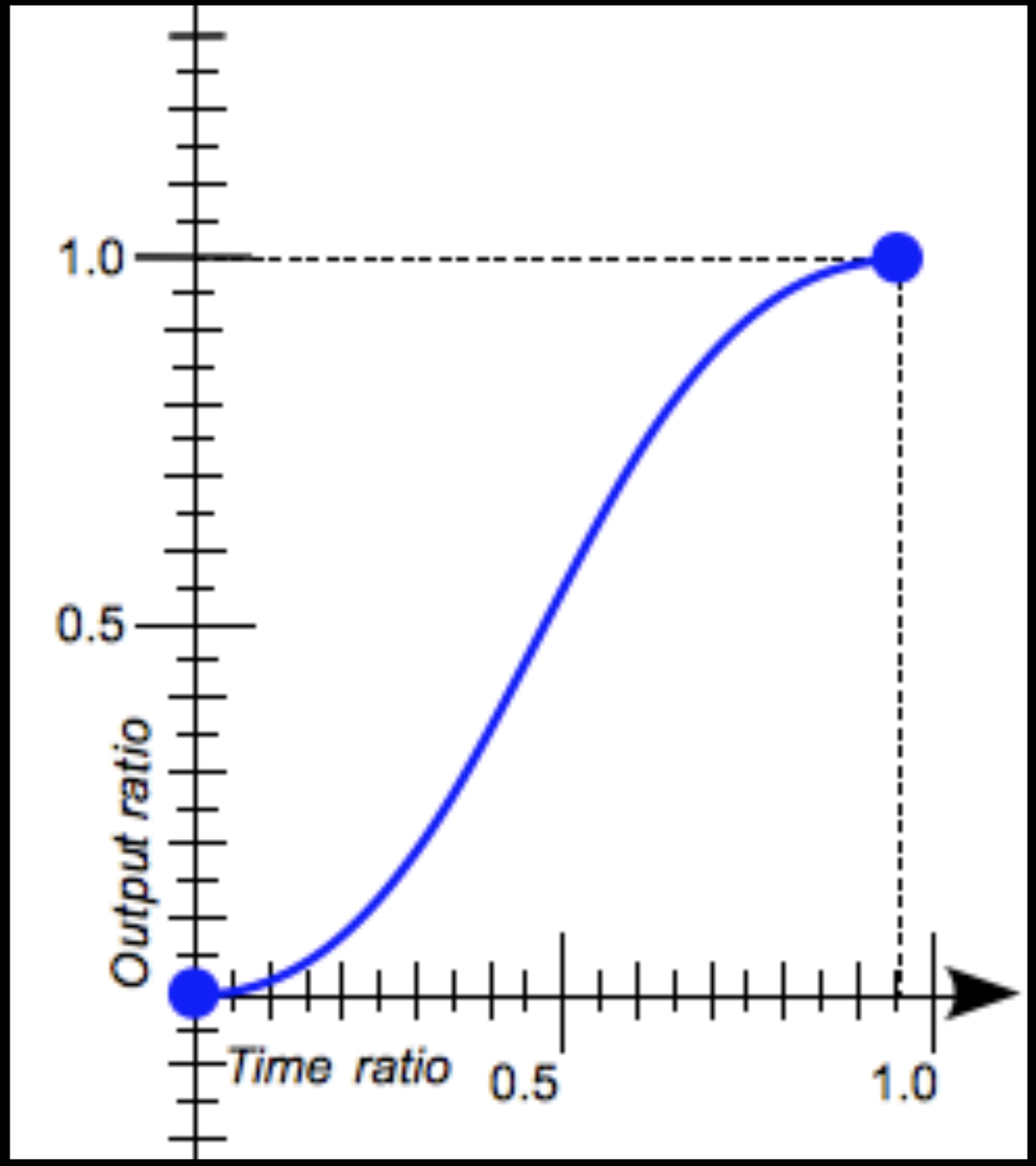


`ease-out`

Starts quickly

Slows progressively down to a gentle stop

Equivalent to `cubic-bezier(0.0, 0.0, 0.58, 1.0)`

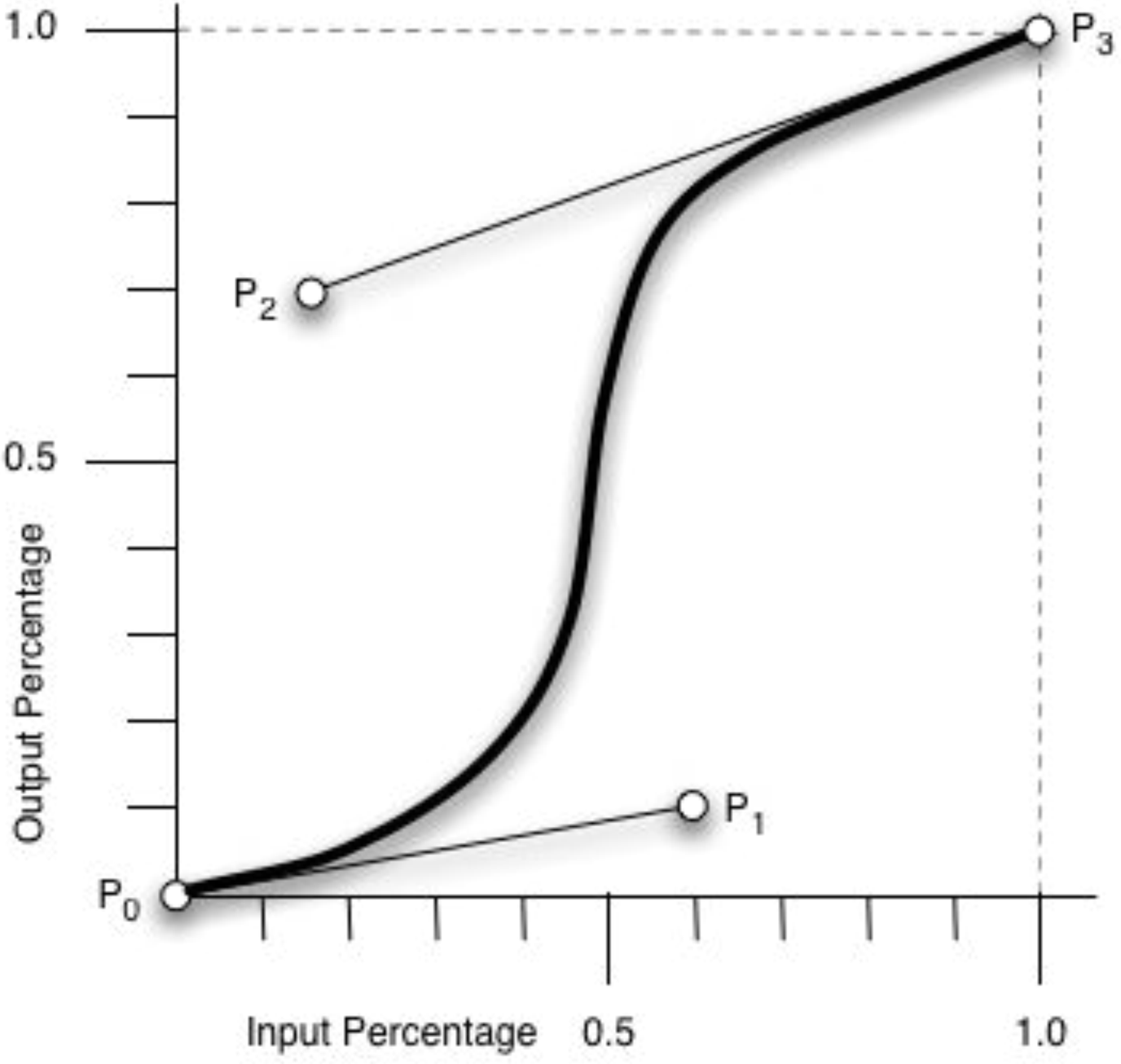


`ease-in-out`

Starts slowly

Accelerates then slows when approaching end to a gentle stop

Equivalent to `cubic-bezier(0.42, 0.0, 0.58, 1.0)`


1.0
0.5
Output Percentage
P0
P1
P2
P3
Input Percentage
0.5
1.0

`cubic-bezier(x1, y1, x2, y2)`

Defines a *cubic Bézier curve*, which is defined by 4 points:

» `P0`: curve's initial time & state (always `0, 0` in CSS)
» `P1`: (defined by `x1` & `y1`)
» `P2`: (defined by `x2` & `y2`)
» `P3`: curve's final time & state (always `1, 1` in CSS)

`x` must be a `<number>` between `0` & `1`

`y` must be a `<number>` (if outside `0`-`1`, you may get a *bouncing effect*)

Ceaser CSS EASING ANIMATION TOOL

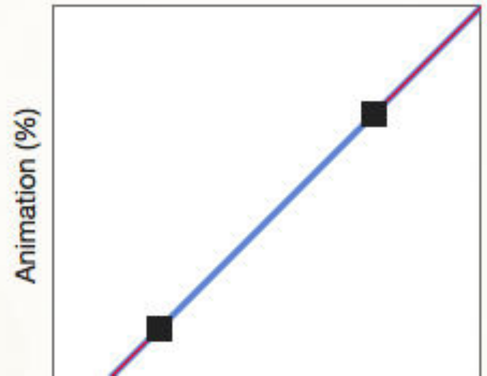
Tweet

1. Choose an easing type and test it out with a few effects.
2. If you don't quite like the easing, grab a handle and fix it.
3. When you're happy, snag your code and off you go.

Now that we can use CSS transitions in all the modern browsers, let's make them pretty. I love the classic Penner equations with Flash and jQuery, so I included most of those. If you're anything like me*, you probably thought this about the default easing options: "ease-in, ease-out...yawn." The mysterious cubic-bezier has a lot of potential, but was cumbersome to use. Until now. Also, touch-device friendly!

*If you *are* anything like me, we should be friends @matthewlein

Note: Bugfixes have landed, so the newest Webkit now supports values above 1 and below 0. For the time being, I am including fallback code for older Webkit that is clamped between 0 and 1 when needed.



Easing: linear

Duration: 500

Effect: Left Width Height Opacity

Ceaser CSS EASING ANIMATION TOOL

Tweet

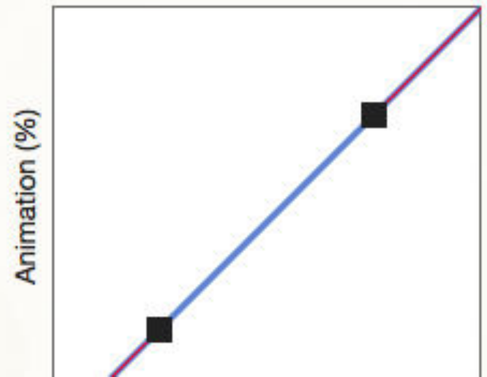
1. Choose an easing type and test it out with a few effects.
2. If you don't quite like the easing, grab a handle and fix it.
3. When you're happy, snag your code and off you go.

Now that we can use CSS transitions in all the modern browsers, let's make them pretty. I love the classic Penner equations with Flash and jQuery, so I included most of those. If you're anything like me*, you probably thought this about the default easing options: "ease-in, ease-out...yawn." The mysterious cubic-bezier has a lot of potential, but was cumbersome to use. Until now. Also, touch-device friendly!

*If you *are* anything like me, we should be friends @matthewlein

Note: Bugfixes have landed, so the newest Webkit now supports values above 1 and below 0. For the time being, I am including fallback code for older Webkit that is clamped between 0 and 1 when needed.

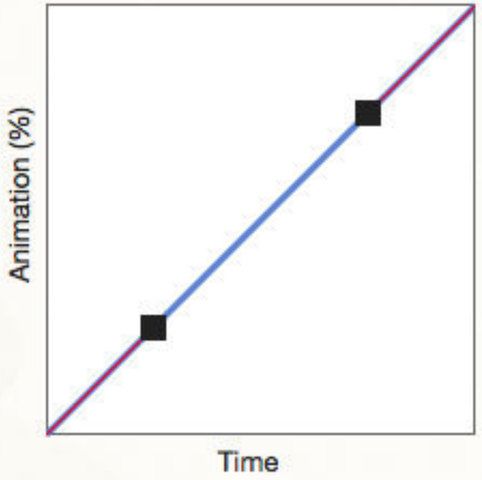
matthewlein.com/ceaser/



Easing: linear

Duration: 500

Effect: Left Width Height Opacity



Easing: linear

Duration: 500

Effect: Left Width Height Opacity

Code snippets, short and long-hand:

```
-webkit-transition: all 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750);
   -moz-transition: all 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750);
     -o-transition: all 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750);
        transition: all 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750); /* linear */

-webkit-transition-timing-function: cubic-bezier(0.250, 0.250, 0.750, 0.750);
   -moz-transition-timing-function: cubic-bezier(0.250, 0.250, 0.750, 0.750);
     -o-transition-timing-function: cubic-bezier(0.250, 0.250, 0.750, 0.750);
        transition-timing-function: cubic-bezier(0.250, 0.250, 0.750, 0.750); /* linear */
```

If this saves you time, or blows your mind, consider making a Donation to keep these projects alive.

HTML

CSS

```
8
9  .linear .box {
10   -webkit-transition-timing-function:
   linear;
11   transition-timing-function: linear;
12 }
13 .ease-in .box {
14   -webkit-transition-timing-function:
   ease-in;
15   transition-timing-function: ease-in;
16 }
17 .ease-out .box {
18   -webkit-transition-timing-function:
   ease-out;
19   transition-timing-function: ease-out;
20 }
21 .ease-in-out .box {
22   -webkit-transition-timing-function:
   ease-in-out;
23   transition-timing-function: ease-in-
   out;
24 }
25
26 #go:target .box {
27   -webkit-transform: translate(300px);
28   transform: translate(300px);
```

JS

Animate Reset

All of the animations below last exactly 2 seconds

Ease (default)

Linear

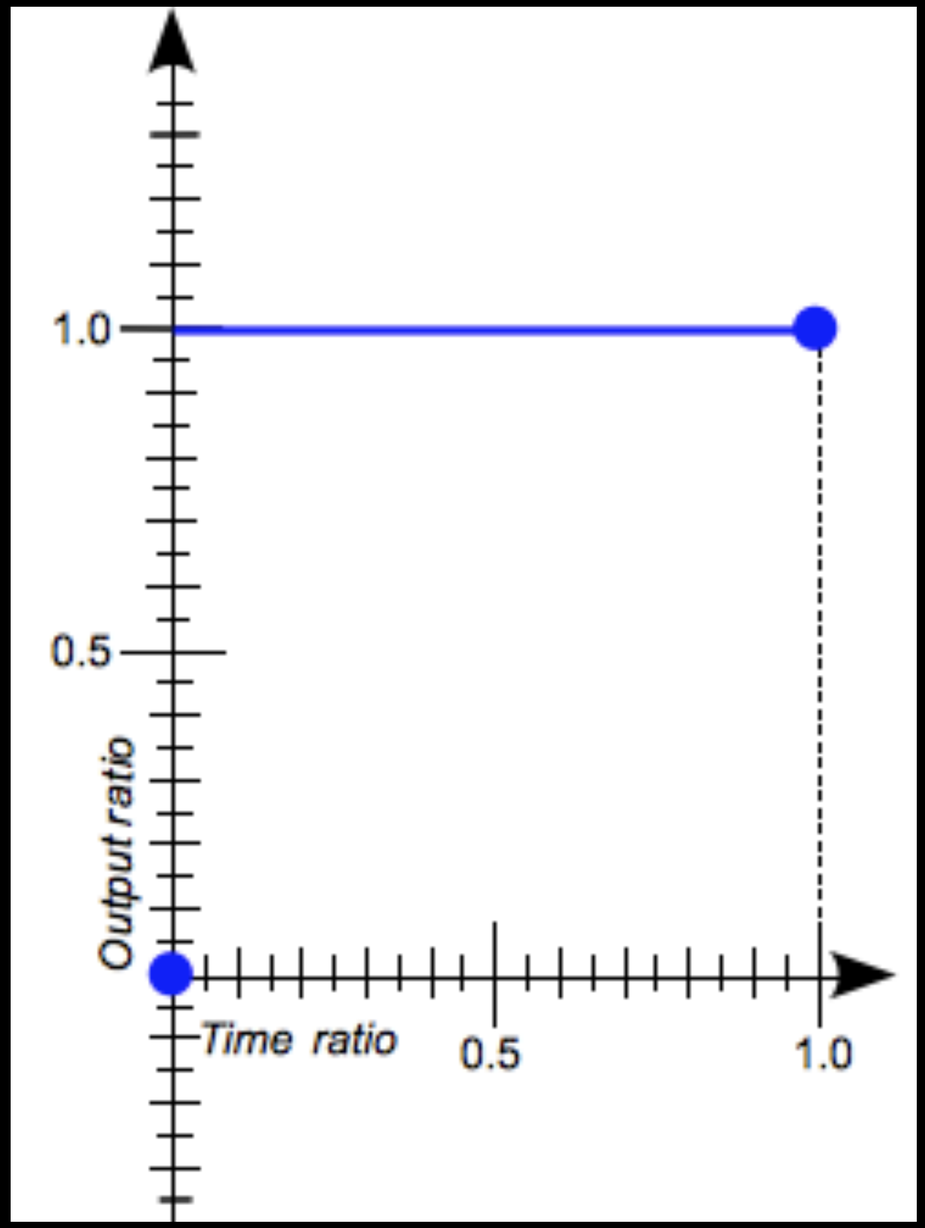
Ease In

Ease Out

Ease In-Out

Values for staircase functions

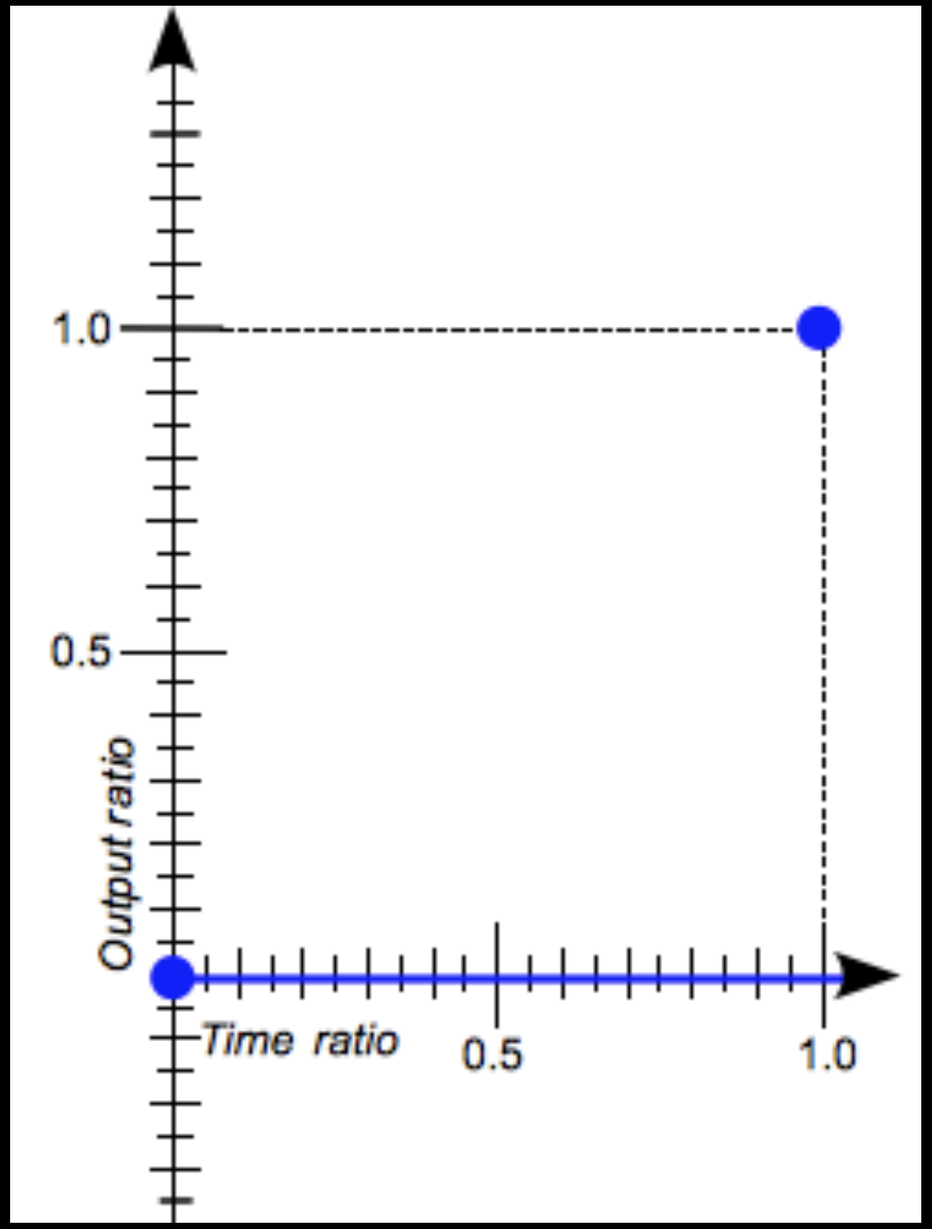
- `step-start`
- `step-end`
- `steps(number-of-steps, direction)`



`step-start`

Animation jumps immediately to the end state & stays until end of animation

Equivalent to `steps(1, start)`



`step-end`

Animation stays in initial state until the end, when it jumps directly to its final position

Equivalent to `steps(1, end)`

`steps(number-of-steps, direction)`

`number-of-steps`: positive `<integer>` representing the amount of equidistant “steps” in the stepping function

`direction`: keyword indicating if the function is left- or right-continuous

2 values

» `start`: *left-continuous* function, so the 1st step happens when the animation *begins*
» `end`: *right-continuous* function, so the last step happens when the animation *ends*

						
`<timing-function>`	10	3.1	4	4	4	2
`cubic-bezier()`	10	8	16	4	Y	—
`steps()`	10	5.1	8	4	4	5

<angle>

`<angle>`

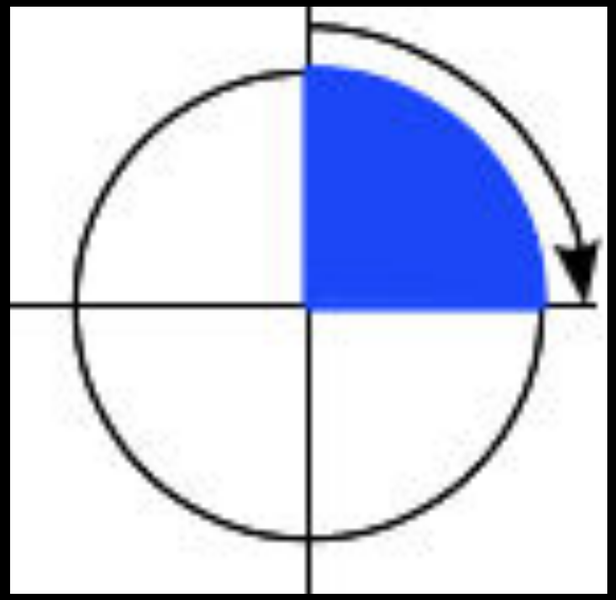
Represents *angle values*: `<number>` data type immediately followed by a unit

Units

» `deg`: degrees (1 full circle is `360deg`)
» `grad`: gradians (1 full circle is `400grad`)
» `rad`: radians (`1rad` is 180/π degrees , so 1 full circle is 2π radians—approximately 6.2832rad)
» `turn`: number of turns (1 full circle is `1turn`)

Positive angles represent right angles, negative angles represent left angles

Right angle



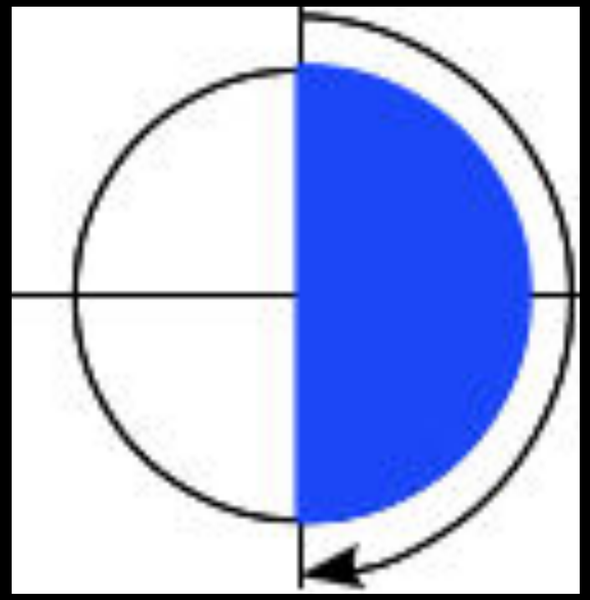
```
90deg

100grad

0.25turn

≈ 1.5708rad
```

Flat angle

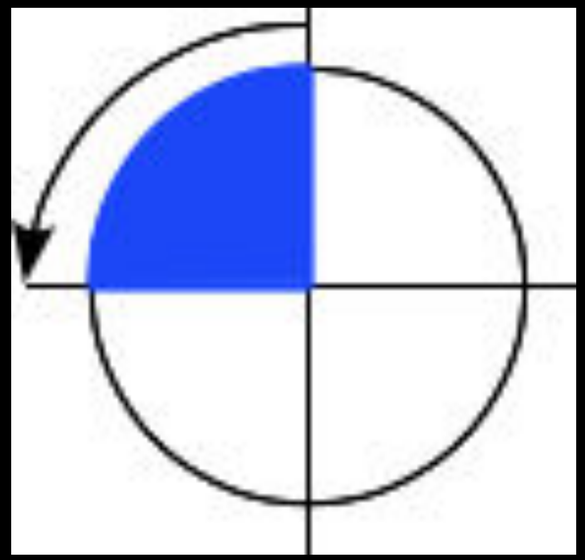


```
180deg

200grad

0.5turn

≈ 3.1416rad
```



Right angle (towards the left)

`-90deg`

`-100grad`

`-0.25turn`

`≈ -1.5708rad`

						
`<angle>`	9	4	2	3.6	?	?

Animation

Animation

Depicting visual change over time

Animation does not have to be motion

Can be:

- `color`
- `position`
- `rotation`
- `border-width`
- `border-radius`
- & many more!
- … but not all

Mozilla Developer Network has a list of animatable properties

Animation is kicked off by an *animation event*

» Page load
» Hover
» Click
» Scrolling
» & much more!

CSS supports 2 kinds of animation

» `transition`: animates styles between 2 *states* (sets of styles)
» `animation`: animates styles between 2 or more *keyframes* (as many as necessary), each with their own state

transition

A transition animates styles *between 2 states*, where *states* are 2 sets of styles

transition-property

transition-duration

transition-timing-function

transition-delay

transition

All 4 properties are part of a transition

```
transition-property: all
transition-duration: 0s
transition-timing-function: ease
transition-delay: 0s
```

If you leave a property set to a default, you don’t need to list the property

`transition-property`

Defines *property or properties to be animated*

`transition-duration`

Defines *how long animation takes* from start to finish

Specified using `<time>` data type

`transition-timing-function`

Defines *acceleration curve* of the animation

Specified using `<timing-function>` data type

`transition-delay`

Amount of *time before animation begins* after the *animation event*

Specified using `<time>` data type

`transition`

Shorthand for `transition-property`, `transition-duration`, `transition-timing-function`, and `transition-delay`

```
transition: width 2s, height 2s, background-color 2s, transform 2s;
```

HTML

CSS

```css
div {
  color: red;
  background-color: yellow;
  border-radius: 0;
  border-color: red;
  text-shadow: none;
}

div:hover {
  color: yellow;
  background-color: black;
  border-radius: 50%;
  border-color: blue;
  text-shadow: 0 0 10px blue, 0 0 15px blue, 0 0 20px blue, 0 0 25px blue;
  transition:
    color 2s ease 0s,
    background-color 1s ease 0s,
    border-radius 3s ease 0s,
    border-color 3s ease 2s,
    text-shadow 5s ease 0s;
}

/* Default transition values */

* {
  transition:
```

JS

HTML

```html
<p>Click to change the ball's top and left positions.</p>
<div id="ball"> </div>
```

CSS

```css
#ball {
  transition:
    all  /* transition-property */
    1s   /* transition-duration */
    ease /* transition-timing-function */
    0s   /* transition-delay */
}
```

JS

```js
var f = document.getElementById('ball');
document.addEventListener('click', function (move) {
  f.style.left = (move.clientX - 25) + 'px';
  f.style.top = (move.clientY - 25) + 'px';
}, false);
```

Click to change the ball's top and left positions.

						
`<timing-function>`	10	3.1	4	4	4	2
`cubic-bezier()`	10	8	16	4	Y	—
`steps()`	10	5.1	8	4	4	5

animation

Animates through *keyframes*

Instead of state A to state B, you use *multiple keyframes* (as many as necessary), each with their own state

Keyframes defined by `@keyframes`, a nested group of rulesets

```
@keyframes rainbow {
  0% {
    color: red;
  }
  50% {
    color: yellow;
  }
  100% {
    color: blue;
  }
}
```

animation-name
animation-duration
animation-timing-function
animation-delay
animation-direction
animation-iteration-count
animation-fill-mode

animation-play-state

animation

`animation-name`

`@keyframes` *ruleset name*, defined by developer

`animation-duration`

Defines *how long animation takes* from start to finish

Specified using `<time>` data type

`animation-timing-function`

Defines *acceleration curve* of the animation

Specified using `<timing-function>` data type

`animation-delay`

Amount of *time before animation begins* after the *animation event*

Specified using `<time>` data type

`animation-direction`

Order keyframes are stepped through

Values

» `normal`: play forward each cycle (default)
» `reverse`: play backward each cycle
» `alternate`: reverse direction each cycle, reversing animation steps & timing functions
» `alternate-reverse`: play backward on 1st play-through, then forward on next, then continue to alternate

`animation-iteration-count`

How many times animation runs

Values

» `<number>`: number of times to repeat (`1` is the default)
» `infinite`: repeat forever

`animation-fill-mode`

Specifies which *keyframe to use before & after execution*

Values

» `none`: do not continue styles from 1st or last keyframe (default)
» `forwards`: continue styles from last keyframe after animation
» `backwards`: styles from 1st keyframe used from start of animation event (during `animation-delay`)
» `both`: styles from 1st keyframe used during `animation-delay`, & styles from last keyframe persist after animation

`animation-play-state`

Determines whether an animation is *running or paused*

Can be queried & set by JavaScript

HTML

```html
<div class="field">
  <div class="ball"> </div>
</div>
```

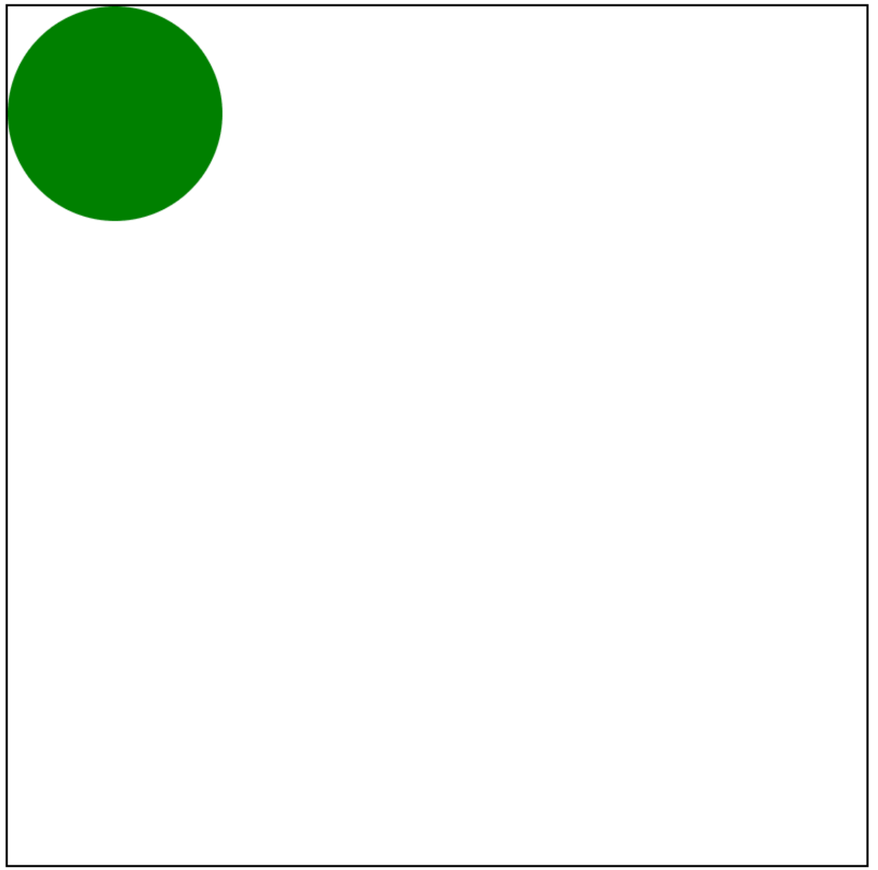
CSS

```css
/* Styles applied before animation
begins */

.field {
  width: 400px;
  height: 400px;
  border: 1px solid black;
  position: relative;
}

.ball {
  width: 100px;
```

JS



						
`<timing-function>`	10	3.1	4	4	4	2
`cubic-bezier()`	10	8	16	4	Y	—
`steps()`	10	5.1	8	4	4	5

Thank you!

scott@granneman.com
www.granneman.com
ChainsawOnATireSwing.com
@scottgranneman

jans@websanity.com
websanity.com

CSS Animation

Visual Change Over Time

R. Scott Granneman

Jans Carton

Changelog

2014-08-04 1.2: Added compatibility charts; added explanations of animations & `transform`; moved `transform` section to “~~Shapes~~Decorating with CSS”
2014-05-15 1.1.1: Clarified some things & fixed others

Licensing of this work