box-sizing property

By default in the CSS box model, the width and height you assign to an element is applied only to the element's content box. If the element has any border or padding, this is then added to the width and height to arrive at the size of the box that's rendered on the screen. This means that when you set width and height, you have to adjust the value you give to allow for any border or padding that may be added. For example, if you have four boxes with width: 25%;, if any has left or right padding or a left or right border, they will not by default fit on one line within the constraints of the parent container.

The box-sizing property can be used to adjust this behavior:

content-box gives you the default CSS box-sizing behavior. If you set an element's width to 100 pixels, then the element's content box will be 100 pixels wide, and the width of any border or padding will be added to the final rendered width, making the element wider than 100px.

border-box tells the browser to account for any border and padding in the values you specify for an element's width and height. If you set an element's width to 100 pixels, that 100 pixels will include any border or padding you added, and the content box will shrink to absorb that extra width. This typically makes it much easier to size elements.

css grids

Flexbox & CSS Grid

"The basic difference between CSS Grid Layout and CSS Flexbox Layout is that flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows, and columns at the same time. The two specifications share some common features, however, and if you have already learned how to use flexbox, the similarities should help you get to grips with Grid."

MDN

The flex-direction property defines in which direction the container wants to stack the flex items - either flex-direction: row or flex-direction: column. However, by using flex-wrap property. Read all about **CSS Flexbox @ W3**.

CSS Grid

A grid is an intersecting set of horizontal and vertical lines - one set defining columns and the other rows. Elements can be placed onto the grid, respecting these column and row lines.

How Grid Layout Works

The process for using the CSS Grid Layout Module is fundamentally simple:

+ Use the display property to turn an element into a grid container. The element's children automatically become grid items.
+ Set up the columns and rows for the grid. You can set them up explicitly and/or provide directions for how rows and columns should get created on the fly (the css grid is very flexible).
+ Assign each grid item to an area on the grid. If you don't assign them explicitly, they flow into the cells sequentially.

The element that has the display: **grid property** applied to it becomes the grid container and defines the context for grid formatting. All of its direct child elements automatically become grid items that end up positioned in the grid. You can define an explicit grid with grid layout but the specification also deals with the content added outside of a declared grid, which adds additional rows and columns when needed. Features such as adding "as many columns that will fit into a container" are included.

Grid line

The horizontal and vertical dividing lines of the grid are called grid lines.

Grid cell

The smallest unit of a grid is a grid cell, which is bordered by four adjacent grid lines with no grid lines running through it.

Grid area

A grid area is a rectangular area made up of one or more adjacent grid cells.

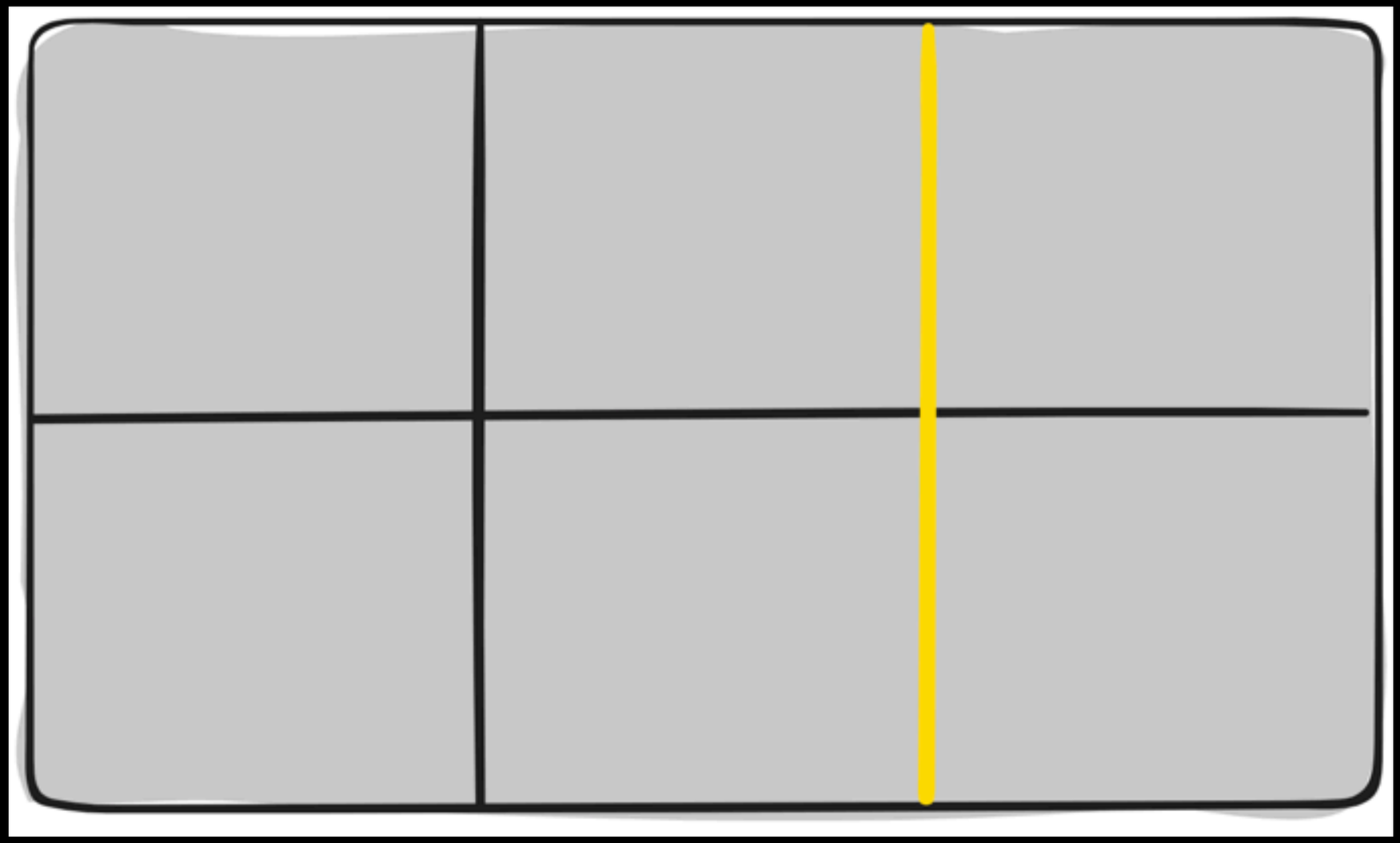
Grid track

The space between two adjacent grid lines is a grid track, which is a generic name for a grid column or a grid row. Grid columns are said to go along the block axis, which is vertical (as block elements are stacked) for languages written horizontally. Grid rows follow the inline (horizontal) axis.

The structure established for the grid is independent from the number of grid items in the container. You could place 4 grid items in a grid with 12 cells, leaving 8 of the cells as 'whitespace.' That's the flexibility of grids. You can also set up a grid with fewer cells than grid items, and the browser adds cells to the grid to accommodate them.

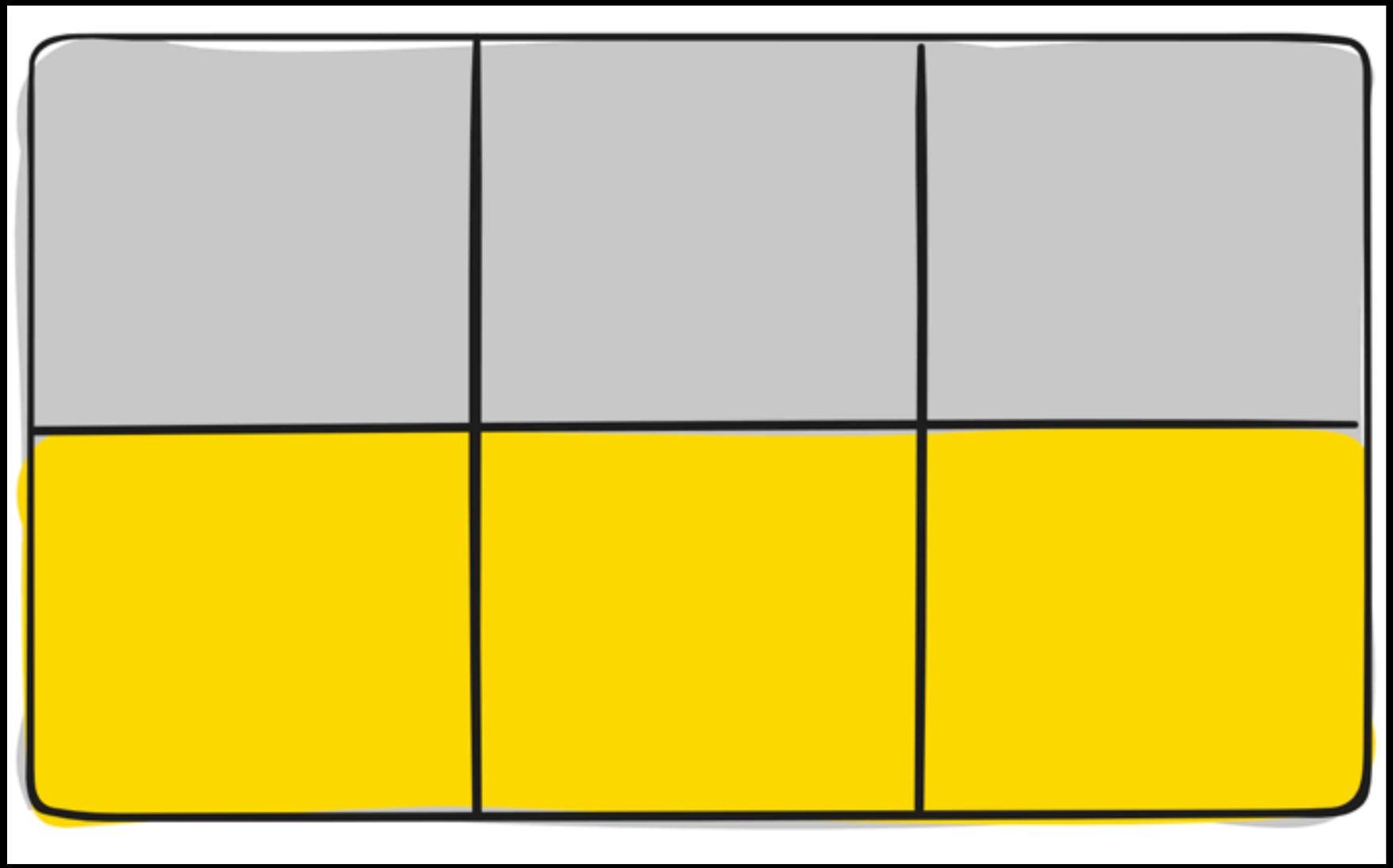
Grid line

The horizontal and vertical dividing lines of the grid are called grid lines. They can be either vertical ("column grid lines") or horizontal ("row grid lines") and reside on either side of a row or column. Here the yellow line is an example of a column grid line.



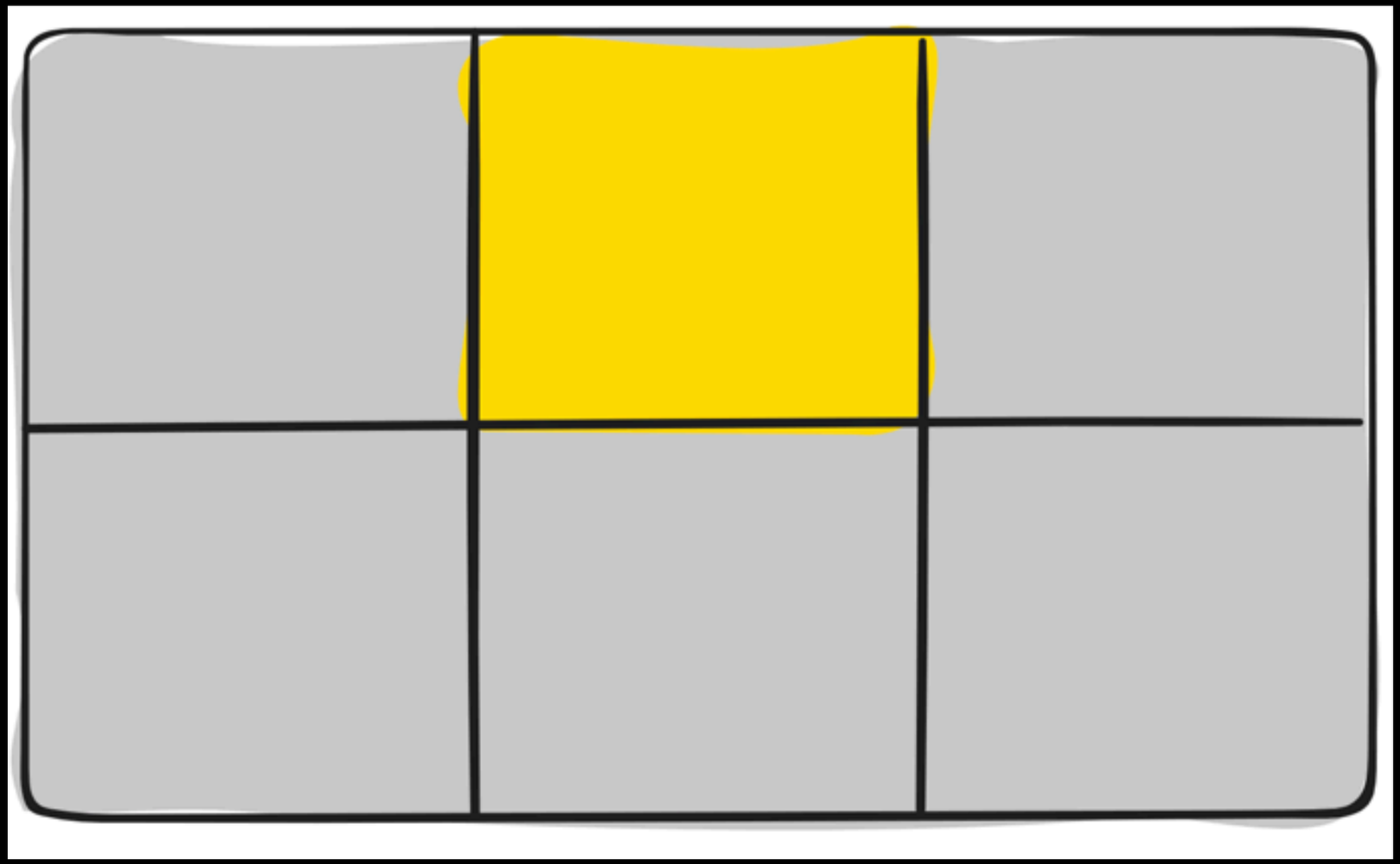
Grid track

The space between two adjacent grid lines is a grid track, which is a generic name for a grid column or a grid row. Grid columns are said to go along the block axis, which is vertical (as block elements are stacked) for languages written horizontally. Grid rows follow the inline (horizontal) axis. Here's the grid track between the second and third row grid lines.



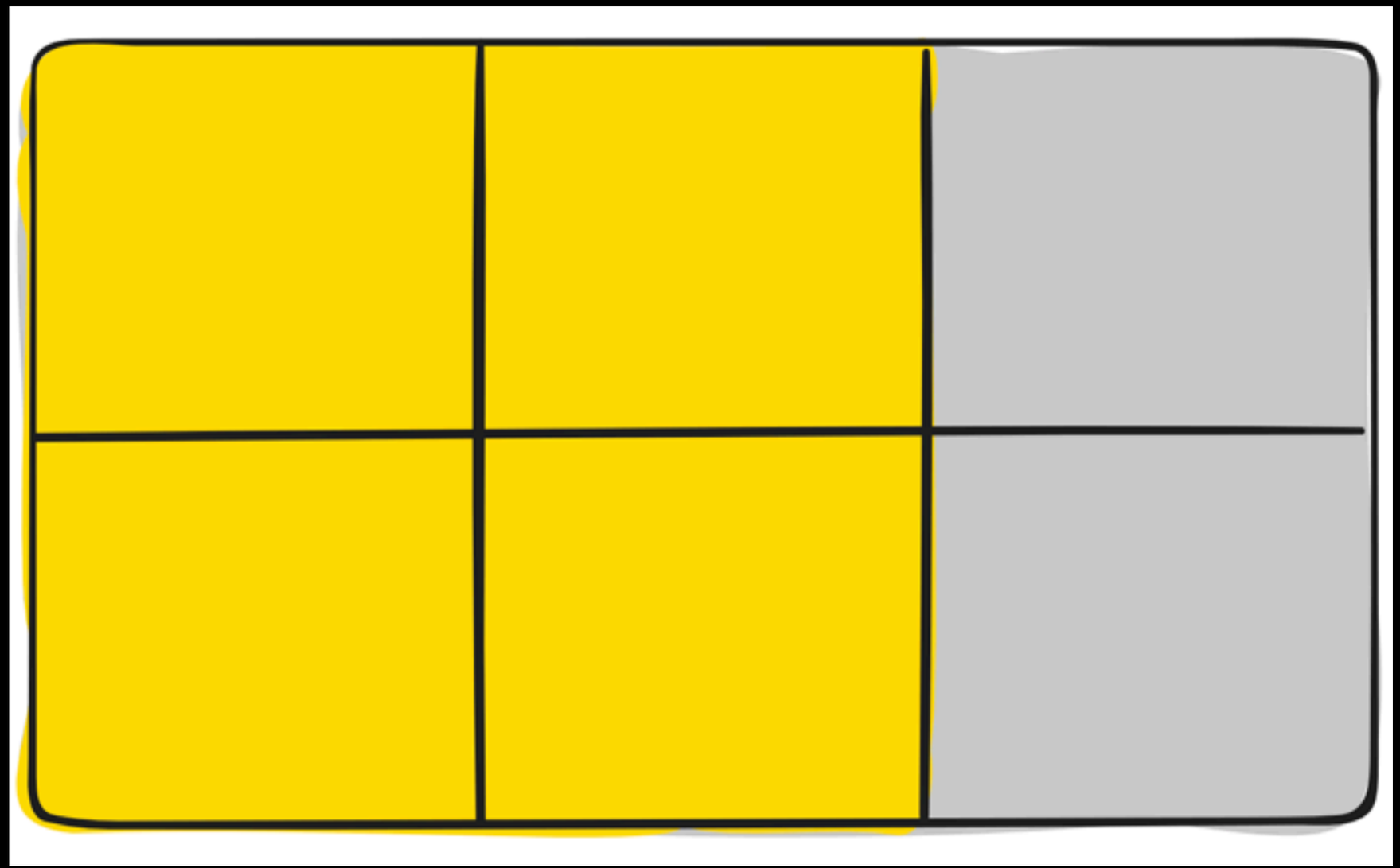
Grid cell

The smallest unit of a grid is a grid cell, which is bordered by four adjacent grid lines with no grid lines running through it. It's a single "unit" of the grid. Here's the grid cell between row grid lines 1 and 2, and column grid lines 2 and 3.



Grid area

A grid area is a rectangular area made up of one or more adjacent grid cells. A grid area may be comprised of any number of grid cells. Here's the grid area between row grid lines 1 and 3, and column grid lines 1 and 3.



Grid Container Properties:

display

grid-template-columns

grid-template-rows

grid-template-areas

grid-template

grid-column-gap

grid-row-gap

grid-gap

justify-items

align-items

place-items

justify-content

align-content

place-content

grid-auto-columns

grid-auto-rows

grid-auto-flow

grid

CSS Tricks w/ links!

Grid Item Properites

grid-column-start

grid-column-end

grid-row-start

grid-row-end

grid-column

grid-row

grid-area

justify-self

align-self

place-self

CSS Grid Functions

repeat ( )

minmax ( )

fit-content ( )

Fr unit

flexible length

image files for the web

SVG

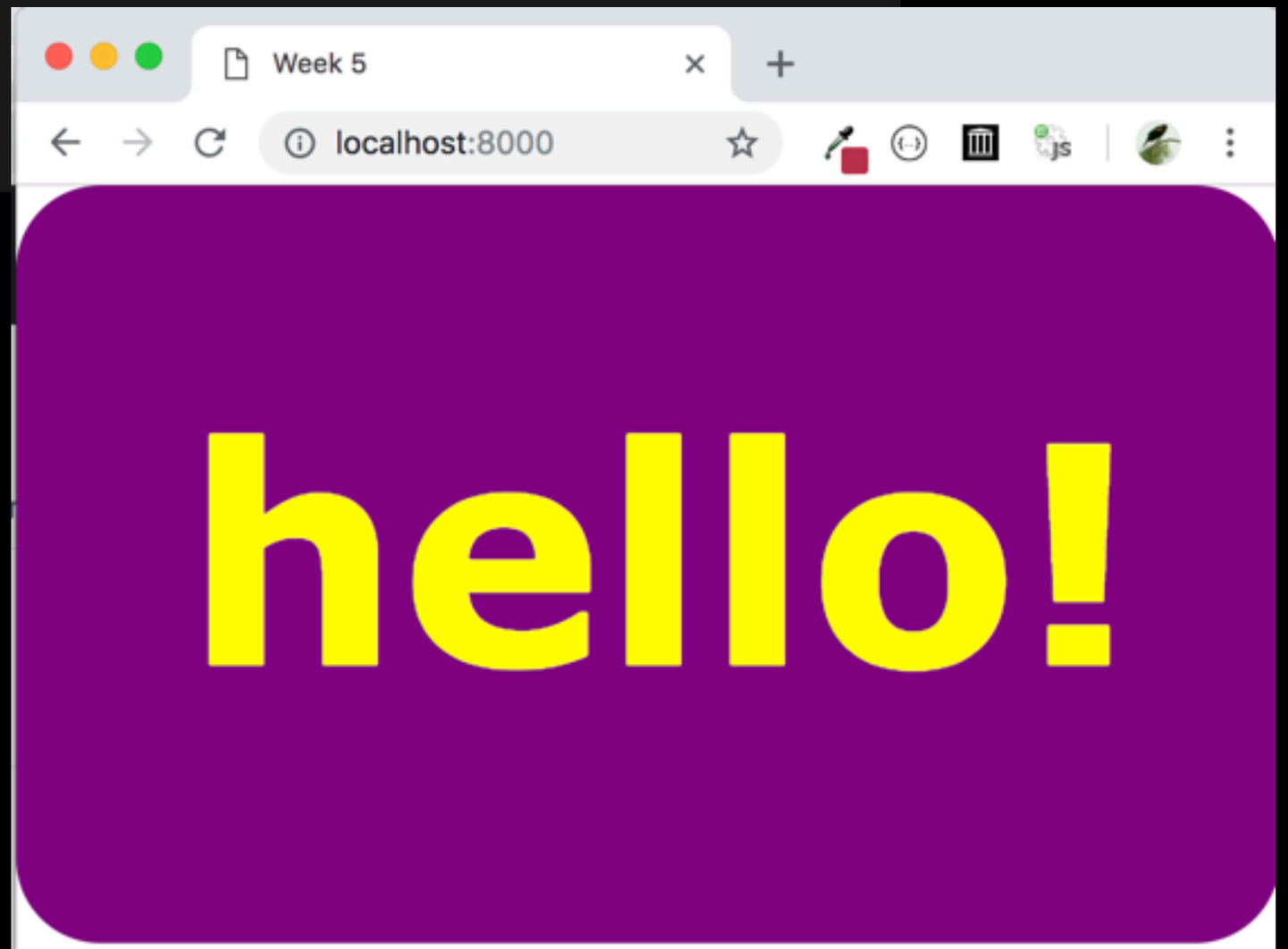
SVG is a 2D vector imaged based on an **XML** (Extensible Markup Language) syntax. It provides the rules and standards for how markup languages should be written and work together. As a result, SVG works well with HTML content.

In SVGs shapes and paths are specified by instructions written out in a text file. Let that sink in: they are images that are written out in text! All of the shapes and paths as well as their properties are written out in the standardized SVG markup language. As HTML has elements for paragraphs **<p>** and navigation **<table>**, SVG has elements that define shapes like rectangle **(rect)**, circle **(circle)**, and paths **(path)**.

A simple example will give you the general idea. Here is the SVG code that describes a rectangle **(rect)** with rounded corners (rx and ry, for x-radius and y-radius) and the word "hello" set as text with attributes for the font and color. Browsers that support SVG read the instructions and draw the image exactly as designed:

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 300 180">
  <rect width="300" height="180" fill="purple" rx="20" ry="20"/>
  <text x="40" y="114" fill="yellow" font-family="'Verdana-Bold'"
  font-size="72">
     hello!
  </text>
</svg>
```

```
rect:hover{
   fill:green;
}
```



This is before responsive web design.

SVG

Advantages of SVGs over bitmapped counterparts for certain image types:

Because they save only instructions for what to draw, they generally require **less data** than an image saved in a bitmapped format. That means faster downloads and better performance.

Because they are **vectors,** they can resize as needed in a responsive layout without loss of quality. An SVG is always nice and crisp. No fuzzy edges.

Because they are text, they integrate well with HTML/XML and can be compressed with tools like Gzip and Brotli, just like HTML files.

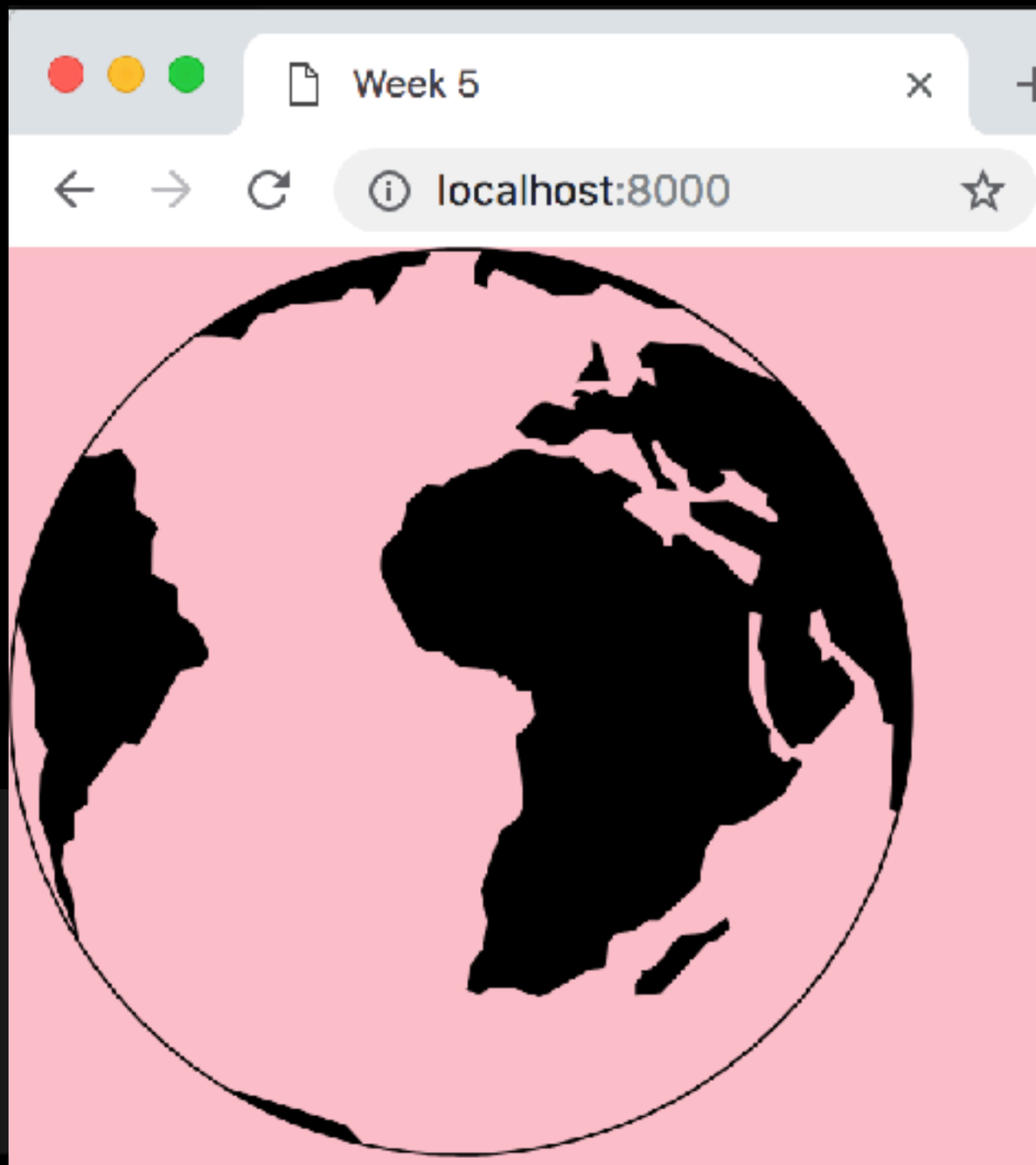
They can be animated.

You can change how they look with CSS.

You can add interactivity with JavaScript so you can add interaction design.

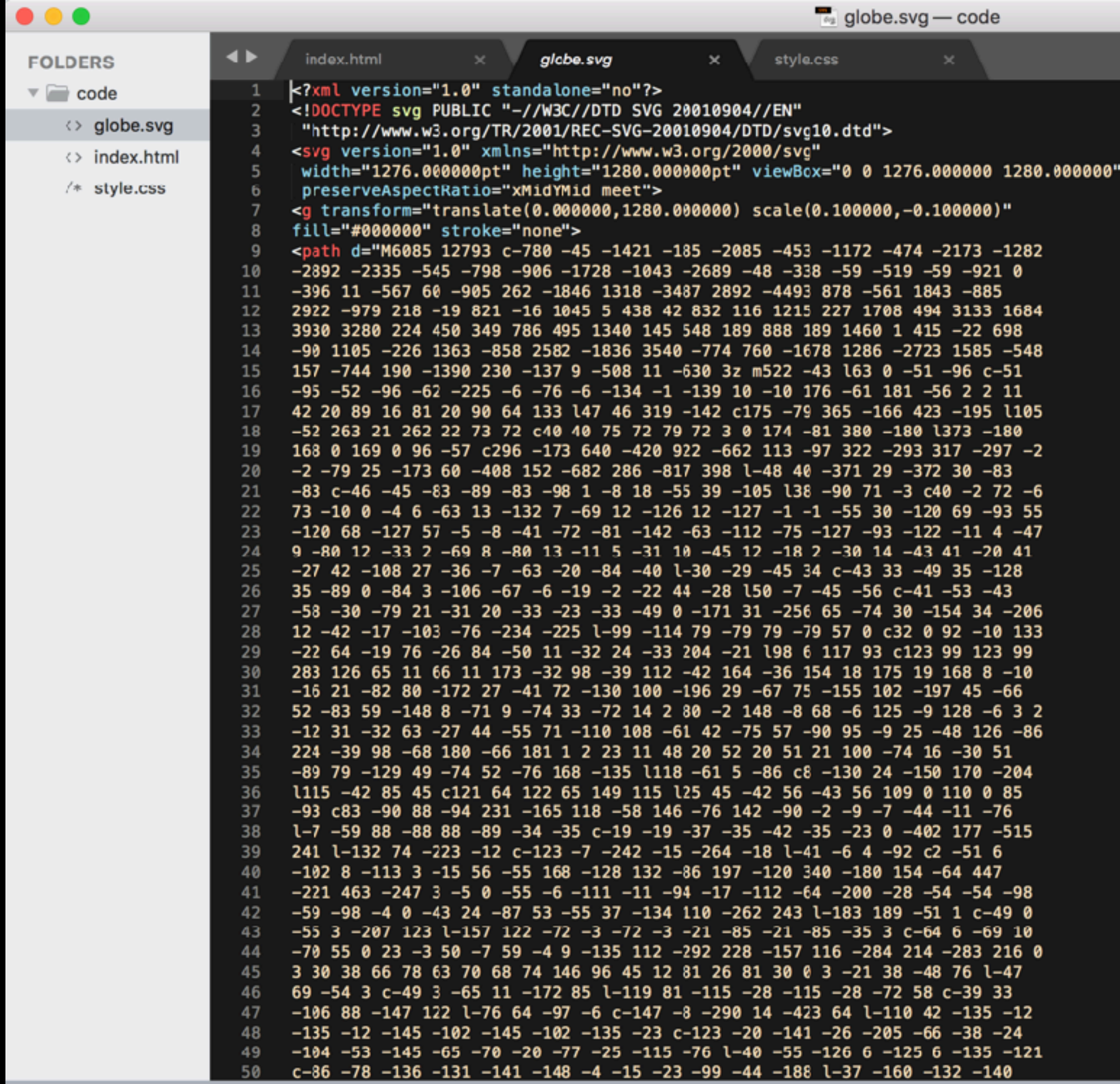
SVG

Embedded with the img element - this will act as a static image.
You can not apply styles, animation or javascript:



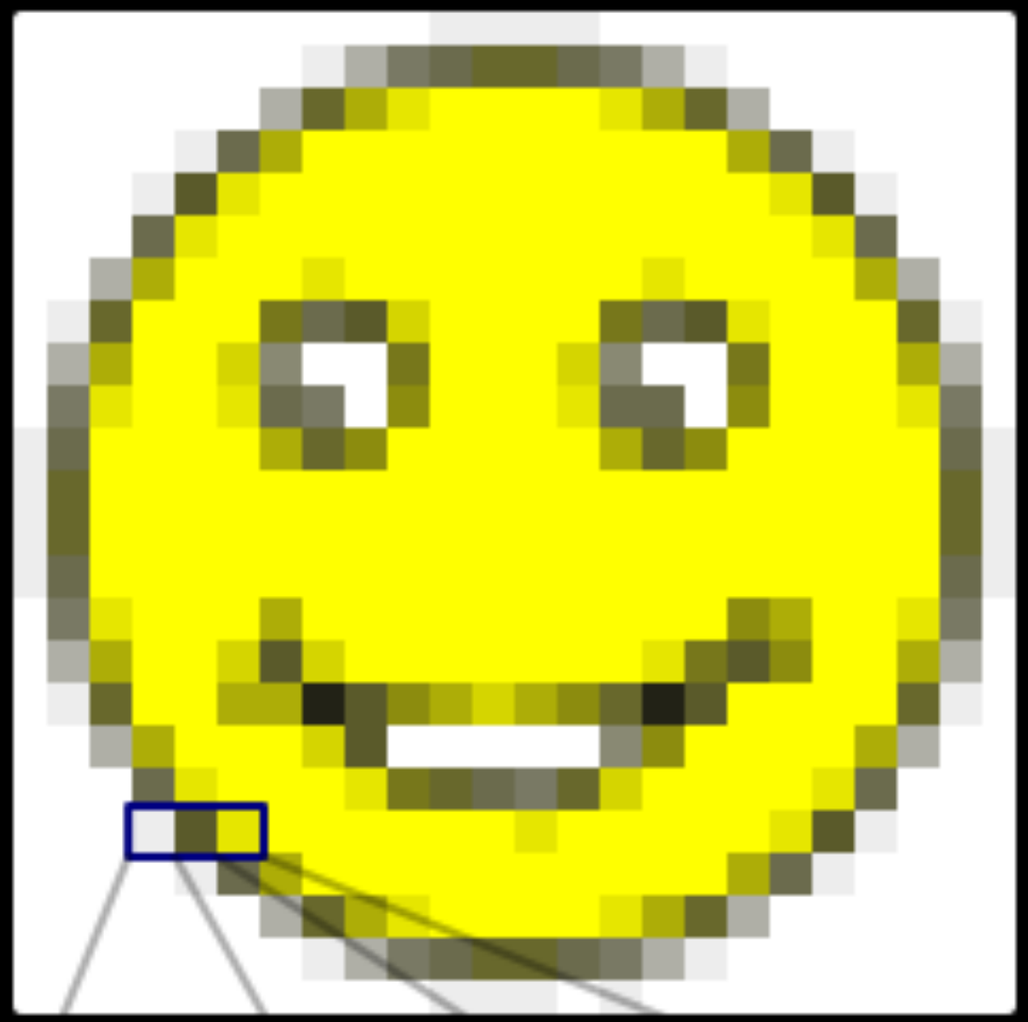

```
<img src="globe.svg" width=300>
```

text editor can
interpret +
display XML



A raster image is a grid of pixels. Each discreet pixel has a red, green, blue + sometimes an alpha (transparency) value.

values: 0 - 255
0 = black
255 = white



rgb (255, 253, 56)

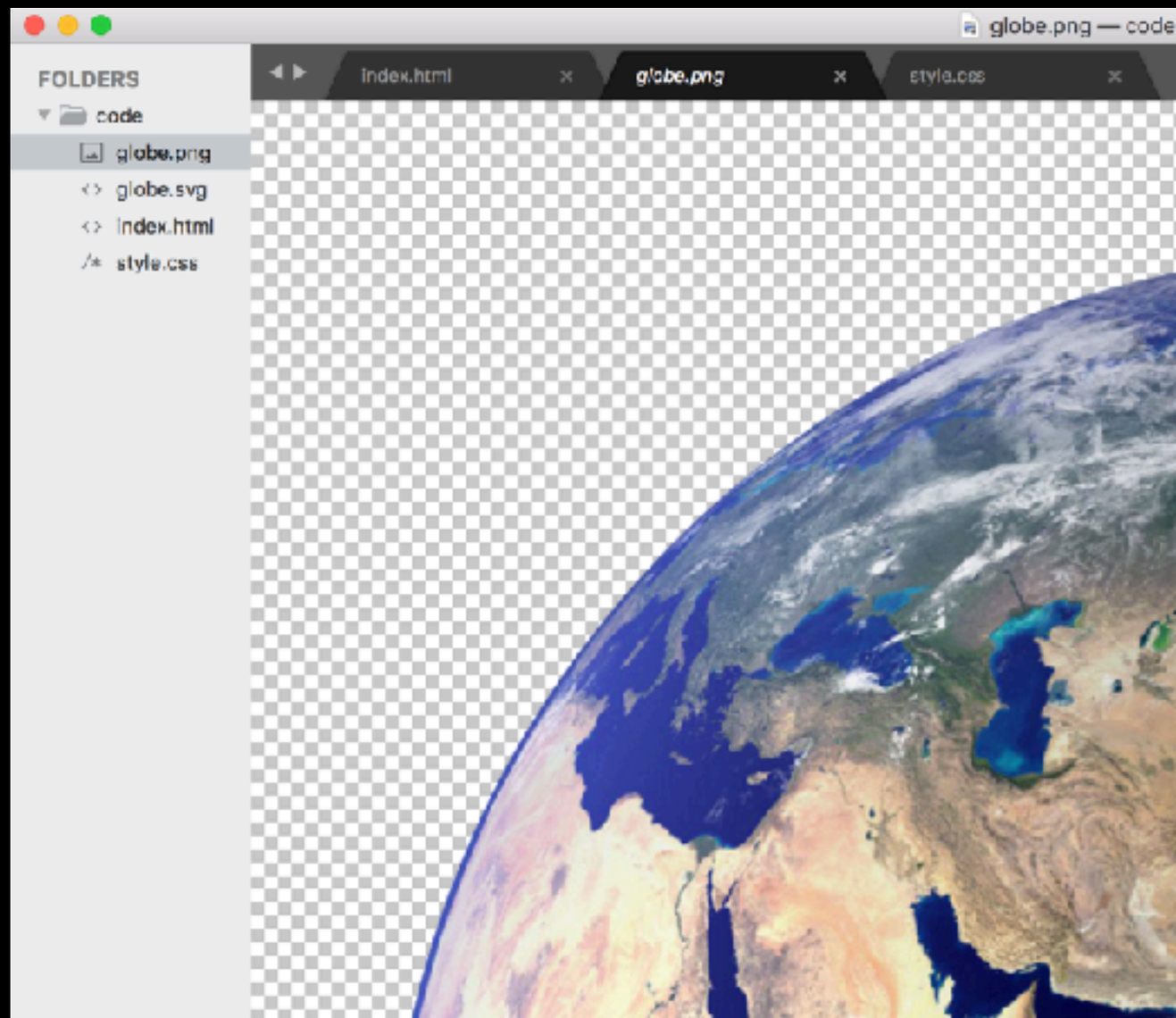
PNG

Portable Network Graphics

is a **raster-**graphics file-format that supports lossless data compression. PNG was developed as an improved, non-patented replacement for Graphics Interchange Format (GIF)

Lempel-Ziv Welch
compression algo
1997, Unisys

text editor (like browser) interprets as image.

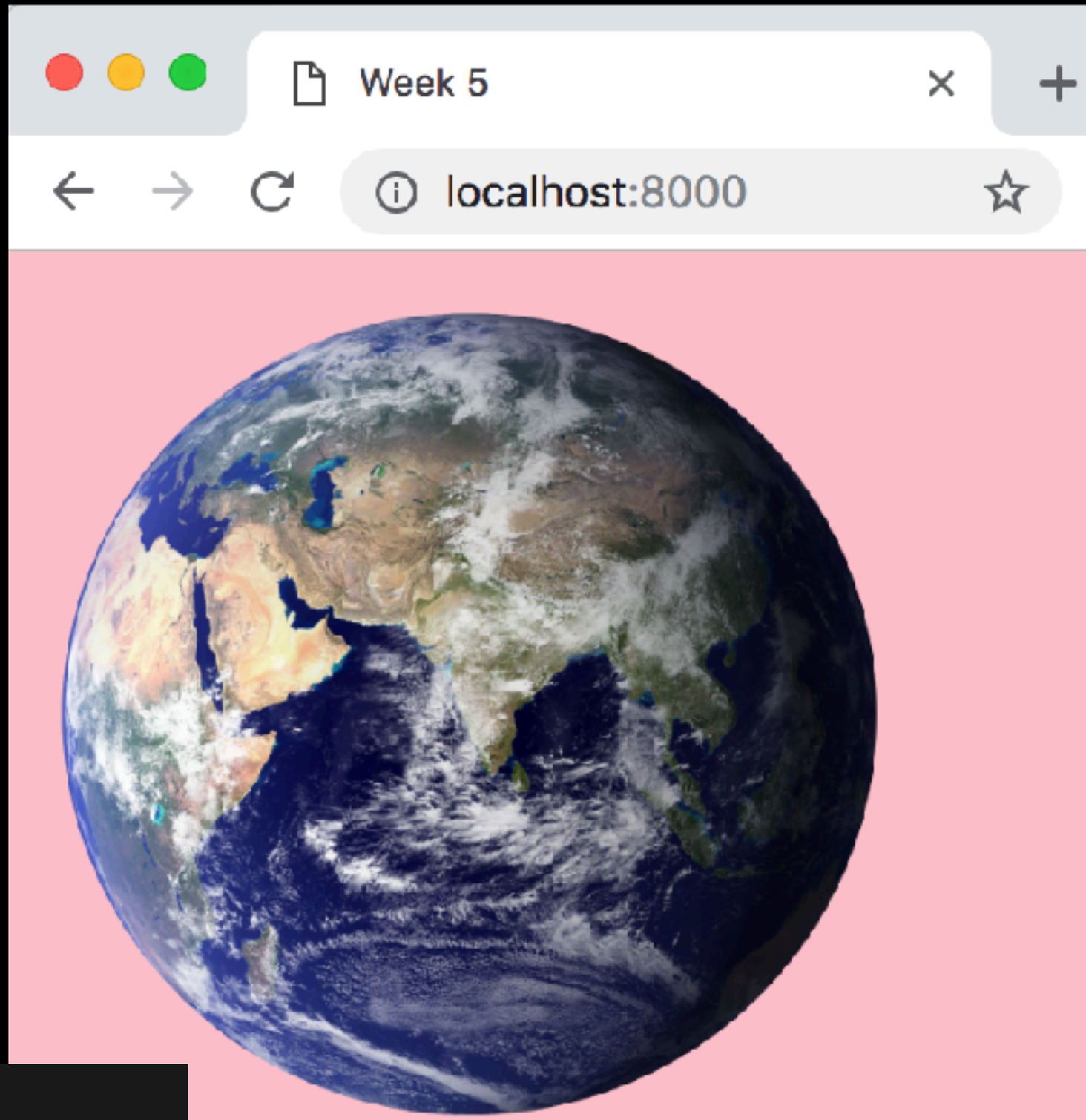


PNG

A raster image is a grid of pixels. Each discreet pixels each have an (R,G,B,A)

red
green
blue
alpha - transparency,
0 -1
in this case it is set to 0.

Using the Canvas we can start to manipulate pixels using Javascript.



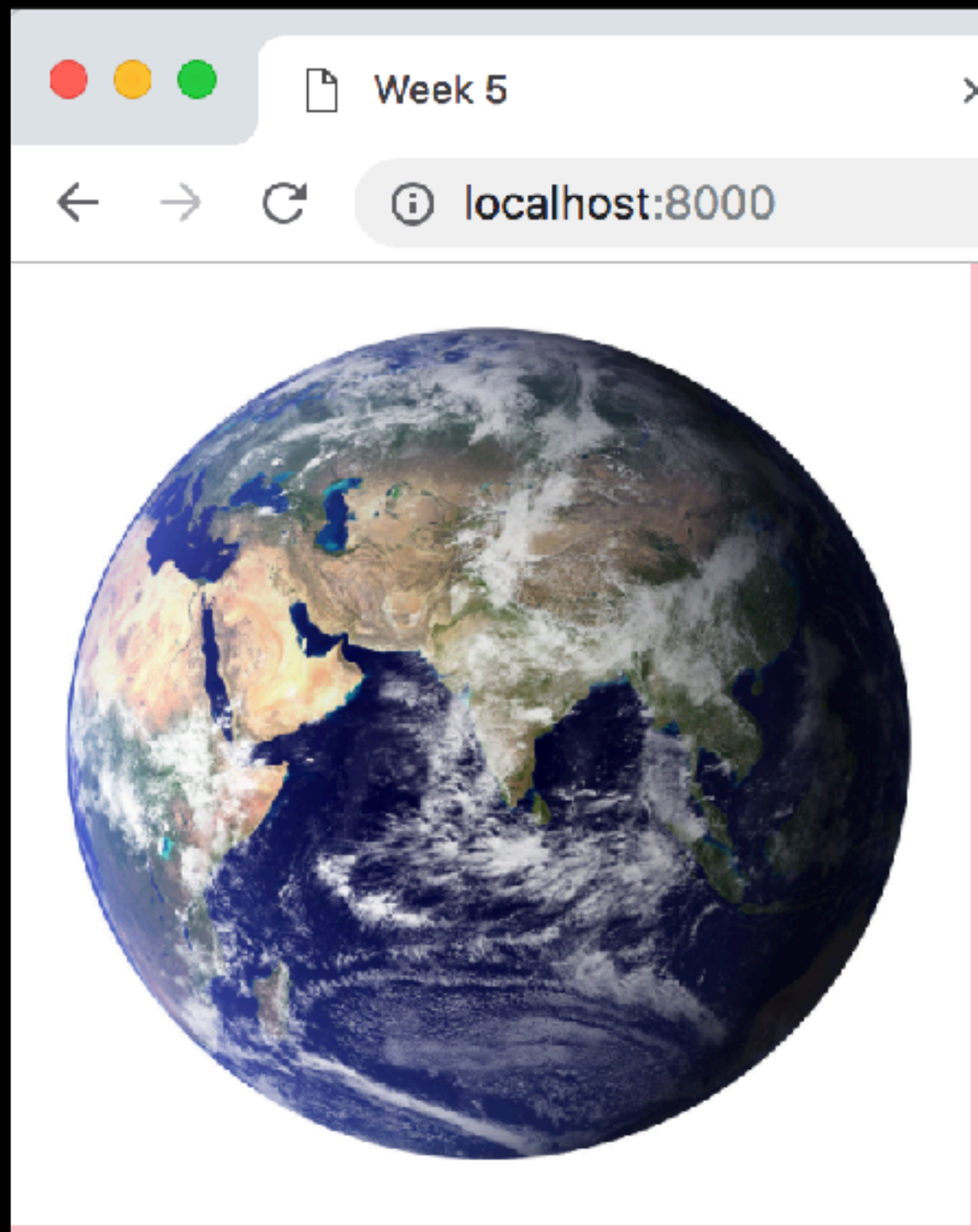

```
<img src="globe.png" width="300">
```

JPG

Joint Photographic Networks Group

a commonly used method of **lossy compression** for digital images, particularly for those images produced by digital photography. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality.

.jpg files have no alpha channel.




```
<img src="globe.jpg" width="300">
```

GIF

Graphic Interchange Format

Steve Whilhite

1987, CompuServe

Words

Enle Li + Liz Xiong

When applying a transition, you have a few decisions to make, each of which is set with a CSS property:

- Which CSS property to change (**transition-property**) (Required)
- How long it should take (**transition-duration**) (Required)
- The manner in which the transition accelerates (**transition-timing- function**)
- Whether there should be a pause before it starts (**transition-delay**)

Transitions require a **beginning state** and an **end state**. The element as it appears when it first loads is the beginning state. The end state needs to be triggered by a state change such as :hover, :focus, or :active…

CSS Animation Selectors

: hover

: focus

: active

transition-property

identifies the CSS property that is changing and that you want to transition smoothly. In our example, it's the background-color. You can also change the foreground color, borders, dimensions, font- and text-related attributes, and many more. TABLE 18-1 lists the animatab CSS properties as of this writing. The general rule is that if its value is a color, length, or number, that property can be a transition property.

Backgrounds

background-color
background-position

Borders and outlines

border-bottom-color
border-bottom-width
border-left-color
border-left-width
border-right-color
border-right-width
border-top-color
border-top-width
border-spacing
outline-color
outline-width

Color and opacity

color
opacity
visibility

Font and text

font-size
font-weight
letter-spacing
line-height
text-indent
text-shadow
word-spacing
vertical-align

Position

top
right
bottom
left
z-index
clip-path

Transforms

transform
transform-origin

Element box measurements

height
width
max-height
max-width
min-height
min-width
margin-bottom
margin-left
margin-top
padding-bottom
padding-left
padding-right
padding-top

Animateable CSS Properties

Timing Functions

```
. thisAwesomeClass {
        transition-timing-function :
                ease
                linear
                ease-in
                ease-out
                ease-in-out
                step-start
                step-end
                steps
                cubic-bezier(#,#,#,#)

}
```

the css property

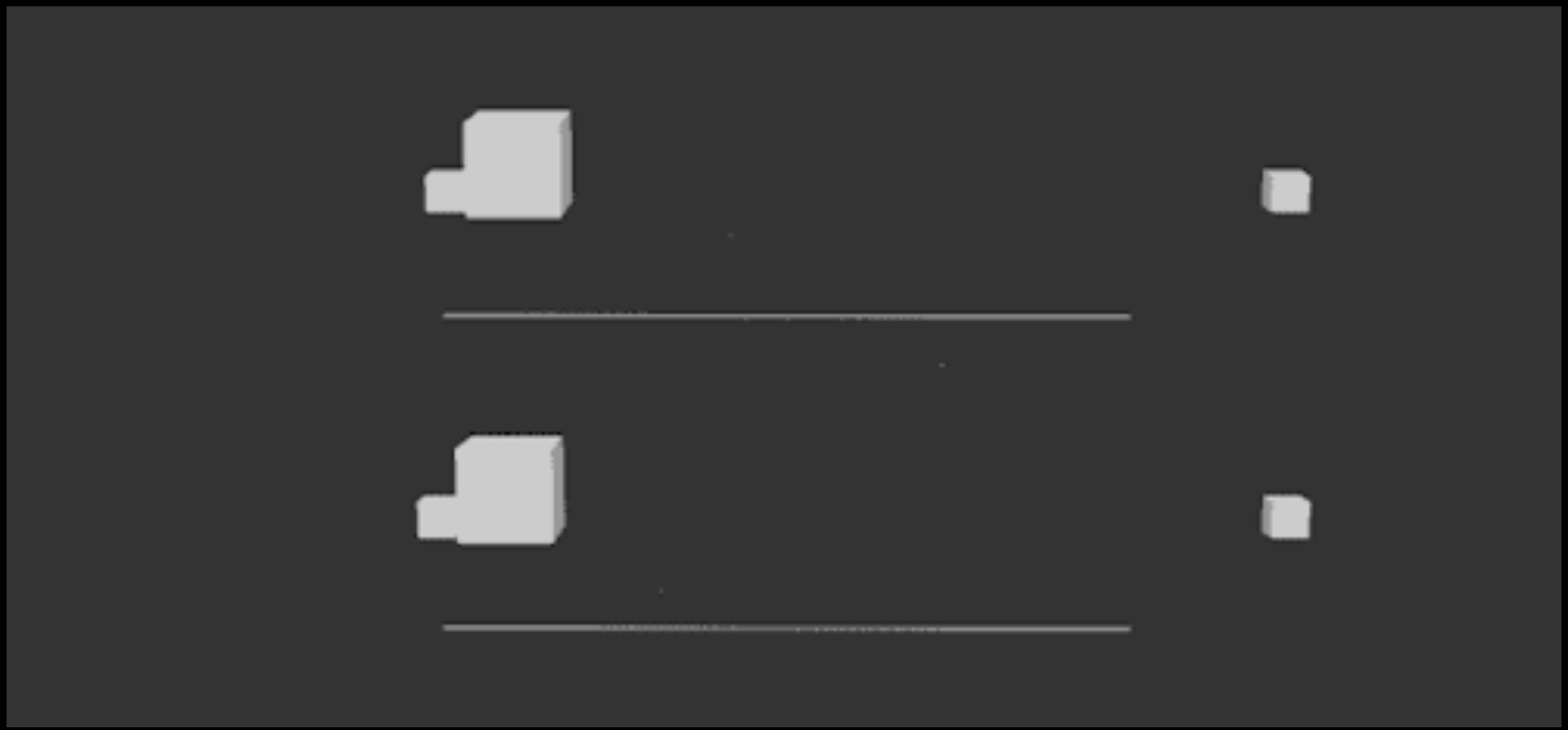
possible values you can set

The **property** and the **duration** are required and form the foundation of a transition, but you can refine it further. There are a number of ways a **transition** can roll out over time.

For example, it could start out fast and then slow down, start out slow and speed up, or stay the same speed all the way through, just to name a few possibilities. I think of it as the transition "style," but in the spec, it is known as the timing function or easing function.

The timing function you choose can have a big impact on the feel and believability of the animation, so if you plan on using transitions and CSS animations, it is a good idea to get familiar with the options.

Animation Principle #6 - Slow (Ease) In + Slow (Ease) Out



Animation Principle #9 - Timing

ease

Starts slowly, accelerates quickly, and then slows down at the end. This is the default value and works just fine for most short transitions.

linear

Stays consistent from the transition's beginning to end. Because it is so consistent, some say it has a mechanical feeling.

ease-in

Starts slowly, then speeds up.

ease-out

Starts out fast, then slows down.

ease-in-out

Starts slowly, speeds up, and then slows down again at the very end. It is similar to ease, but with less pronounced acceleration in the middle.

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

The acceleration of a transition can be plotted with a curve called a Bezier curve. The steep parts of the curve indicate a fast rate of change, and the flat parts indicate a slow rate of change.

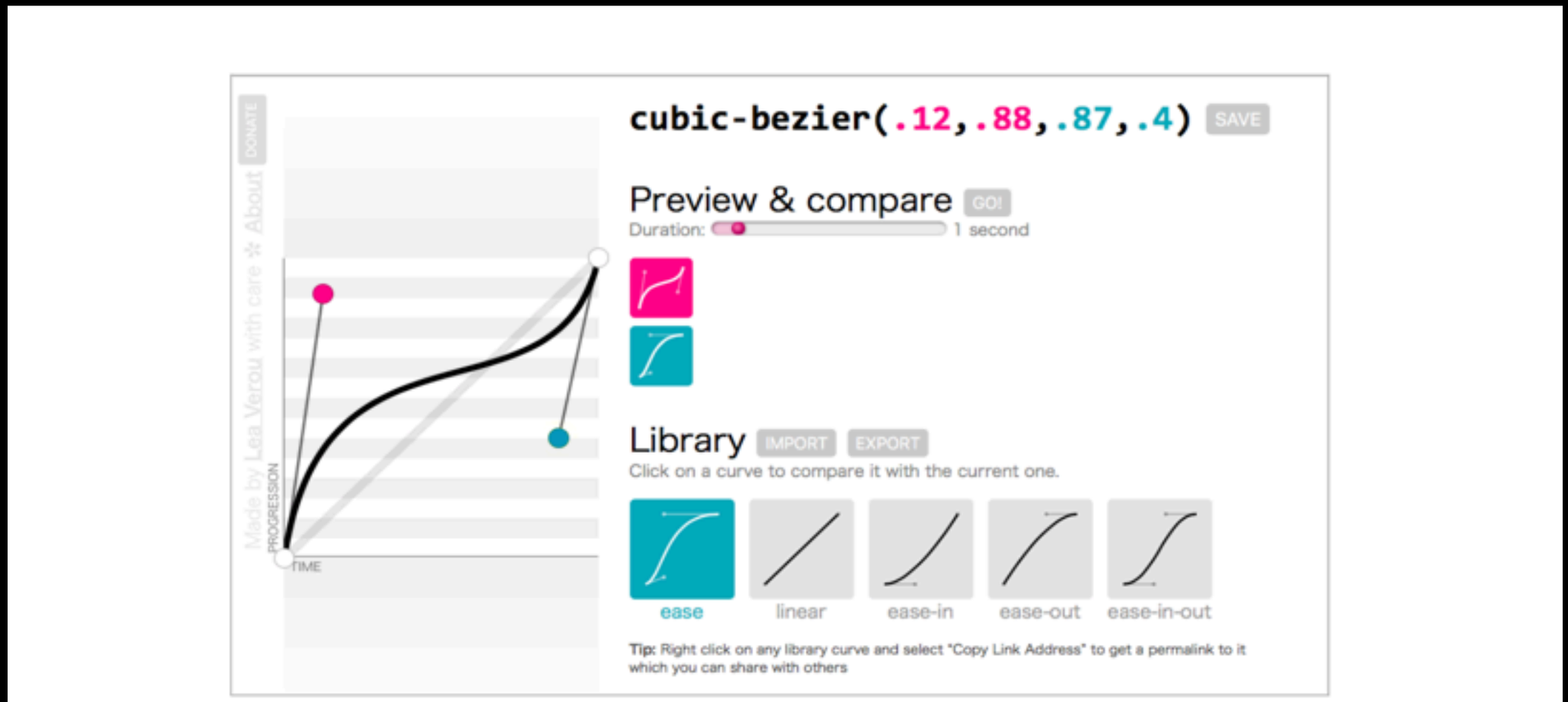


FIGURE 18-2. Examples of Bezier curves from Cubic-Bezier.com. On the left is my custom curve that starts fast, slows down, and ends fast.

You can see that the ease curve is a tiny bit flat in the beginning, gets very steep (fast), then ends flat (slow). The linear keyword, on the other hand, moves at a consistent rate for the whole transition.

You can get the feel of your animation just right by creating a custom curve. The site Cubic-Bezier.com is a great tool for playing around with transition timing and generating the resulting code. The four numbers in the value represent the x and y positions of the start and end Bezier curve handles (the pink and blue dots.

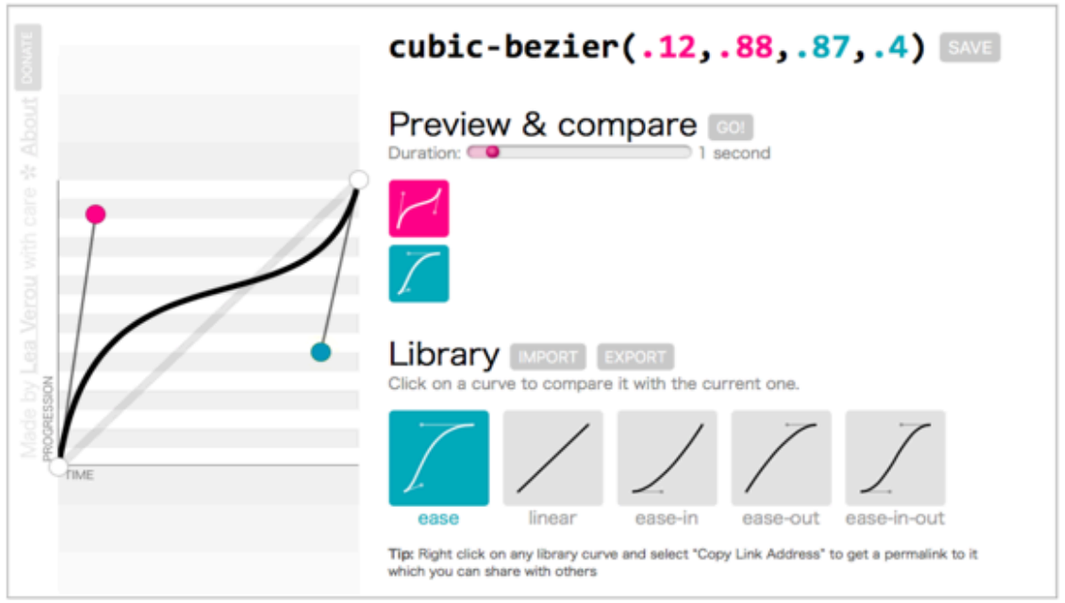


FIGURE 18-2. Examples of Bezier curves from Cubic-Bezier.com. On the left is my custom curve that starts fast, slows down, and ends fast.

steps(#, start|end)

Divides the transitions into a number of steps as defined by a stepping function. The first is the number of steps, and the **start** and **end** keywords define whether the change in sta happens at the begin- ning (**start**) or end of each step. Step animation is especially useful keyframe animation with sprite images. For a better explanation and examples, I recomm the article “Using Multi-Step Animations and Transitions,” by Geoff Graham on CSS-Tricks (*css-tricks.com/using-multi- step-animations-transitions/*).

step-start

Changes states in one step, at the beginning of the duration time (the same as **steps(1,st** The result is a sudden state change, the same as if no transition had been applied at all.

step-end

Changes states in one step, at the end of the duration time (the same as **steps(1,end)**).

transition-delay

The transition-delay property, as you might guess, delays the start of the animation by a specified amount of time.

The Shorthand transition Property

The authors of the CSS3 spec had the good sense to give us the shorthand transition property to combine all of these properties into one declaration.You've seen this sort of thing with the shorthand border property. Here is the syntax:

```
transition: property duration timing-function delay;
```

```
.theClass {
    transition: background-color 0.3s ease-in-out 0.2s;
}
```

The values for each of the **transition-*** properties are listed out, separated by character spaces. The order isn't important as long as the **duration** (which is required) appears before **delay** (which is optional). If you provide only one time value, it will be assumed to be the duration.

The sub-properties of the **animation** property are:

animation-delay	Configures the delay between the time the element is loaded and the beginning of the animation sequence.
animation-direction	Configures whether or not the animation should alternate direction on each run through the sequence or reset to the start point and repeat itself.
animation-duration	Configures the length of time that an animation should take to complete one cycle.
animation-iteration-count	Configures the number of times the animation should repeat; you can specify infinite to repeat the animation indefinitely.
animation-name	Specifies the name of the **@keyframes** at-rule describing the animation's keyframes.

animation-play-state	Lets you pause and resume the animation sequence.
animation-timing-function	Configures the timing of the animation; that is, how the animation transitions through keyframes, by establishing acceleration curves.
animation-fill-mode	Configures what values are applied by the animation before and after it is executing.

@keyframes + animation property