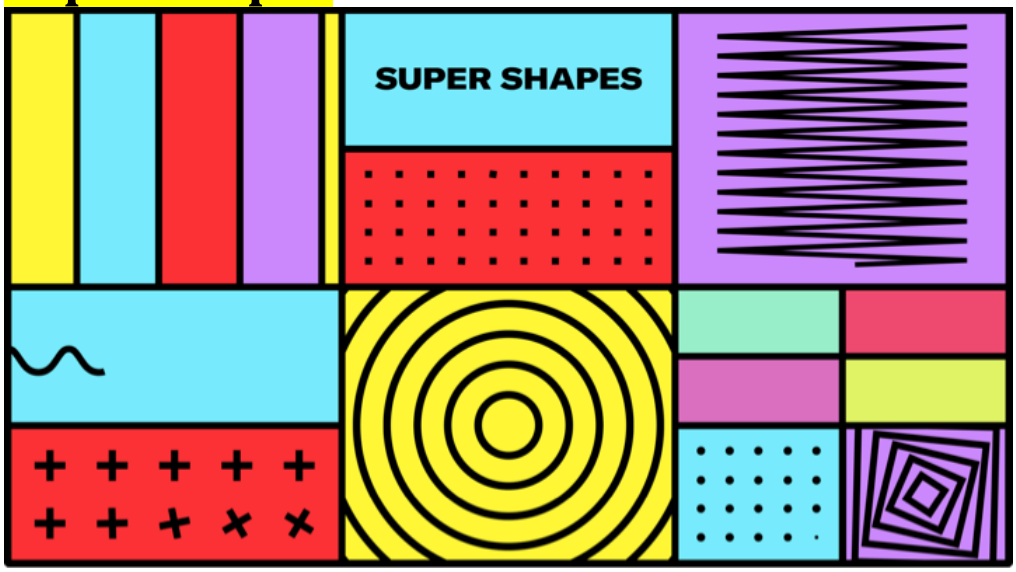
SuperHi-Advanced-Week3

http://guides.superhi.com/advanced/super-shapes/#/

Super Shapes



A dive into web motion graphics and how we can use SVG to create immersive animations.

What we'll learn...

- Using grid systems on the web
- Writing our CSS in a more minimal style
- What SVGs are and how to use them
- Animating SVGs using anime.js

What are SVGs?

SVG stands for `scalable vector graphics`, and it's a vector file format that works great on the web. Due to the fact they're vector files, we can enlarge them to any size without pixelating. This makes them perfect for modern retina screens.

Using SVGs on our page

Both Sketch an Illustrator can export SVG files. We can use SVG files with an `img` tag or using the raw code.

As an image tag

```
<img src="superhi.svg">
```

As raw SVG

```
<svg width="199px" height="199px" viewBox="0 0 199 199" version="1.1"
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
 <defs></defs>
 <g id="Page-1" stroke="none" stroke-width="1" fill="none" fill-rule="evenodd">
  <path d="M141.9585,113.9848 C141.9585,113.9848 134.7065,145.0678
99.4735,145.0678 C64.2435,145.0678 56.9905"></path>
 </g>
</svg>
```

The advantage of using an `img` tag is that it's easy and familiar to us, but it won't let us change our image's properties like color or stroke thickness. However, when using the raw `svg` tag, it embeds the actual vector codes with it, meaning we can tweak and manipulate them from our CSS or Javascript.

Tweaking our SVG properties

When we embed our SVG code, it has a bunch of tags in there such as `path`, `g`, `rect`and `circle`. We can write CSS to directly change properties like `fill`, `dash-offset` and`stroke-width`.

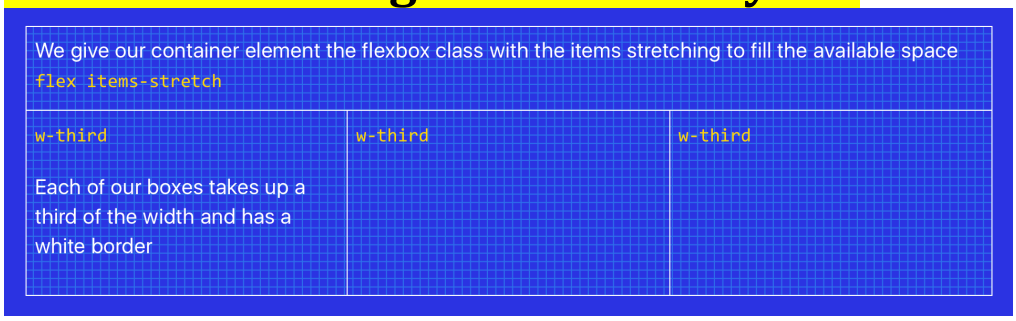
```
.logo path {
 fill: #fc0f3a;
 animation: rainbow 5s infinite;
 animation-fill-mode: forwards;
}
```

Building our grid using Tachyons

http://tachyons.io
Tachyons is a lightweight CSS library that breaks down common CSS properties into reusable classes that we write into our HTML.
It's key benefit is that it allows us to*"create fast loading, highly readable, and 100% responsive interfaces with as little css as possible."*

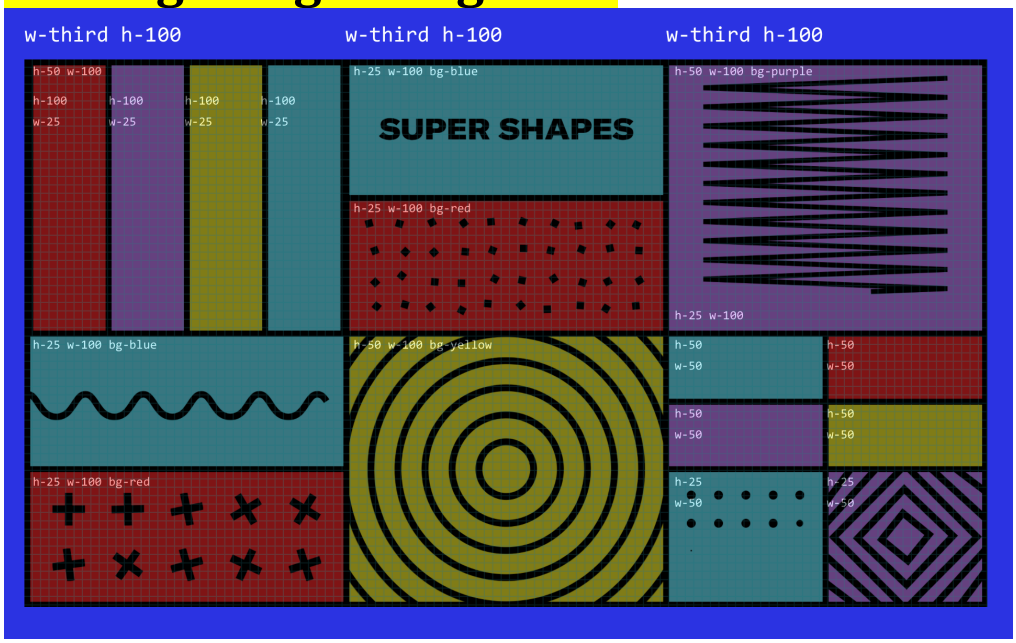
There's lots of really good examples of common web design patterns in the Tachyons components and website galleries.

A three column grid with Tachyons



More information on how to achieve grid layouts in Tachyons is on the excellent documentation page, as well as the flexbox page.

Putting our grid together



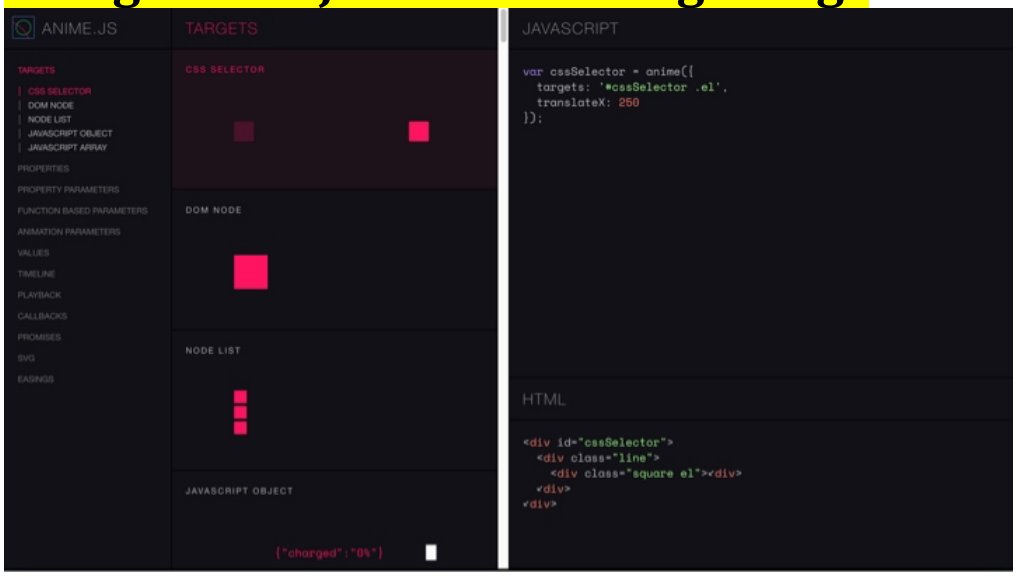
Creating circles in SVG

We can create circles in SVG by copying our SVG code from either Sketch or Illustrator and then modifying it manually.

```
<svg width="400px" height="400px" viewBox="0 0 400 400" version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Created with Sketch.</desc>
  <defs></defs>
  <g stroke="none" stroke-width="1" fill="none" fill-rule="evenodd">
    <circle id="Oval" stroke="#FFFFFF" stroke-width="10" cx="200" cy="200" r="195"></circle>
  </g>
</svg>
```

- `width` our svg width
- `height` our svg height
- `viewBox` first two numbers always 0 and the last two equal to width and height
- `stroke-width` same as in Sketch/Illustrator
- `fill` same as in Sketch/Illustrator
- `cx` circle center x position
- `cy` circle center y position
- `r` radius of the circle

Using anime.js for animating things



anime.js is a lightweight JavaScript animation library. It lets us animate elements on our page with a nice interface and lots of built in helpers. It works on both HTML elements as well as SVG too!

Animating our tunnel

```
anime({
targets: '#tunnel circle',
scale: 1.1,
// we add delay to each one that increments by 50ms
delay: (el, i) => i * 50,
loop: true,
duration: 250,
// animates back from the end to beginning
direction: 'alternate',
// give it some character with an easing fn
easing: 'easeInOutSine'
})

/* tell the circles that we want them to scale from the center */
.tunnel circle {
  transform-origin: center;
}
```

Animating the conveyor belt

```
anime({
  targets: '.conveyor',
  translateX: '-50%',
  duration: 1500,
  loop: true,
  easing: 'linear'
})

.conveyor {
  width: 200%;
}
```

Animating along SVG paths

```
// set up variables for our path
const zigZagPath = document.querySelector('#zigzag path')
// get the length of the path
const zigZagOffset = anime.setDashoffset(zigZagPath)
// set the length of the path
zigZagPath.setAttribute('stroke-dashoffset', zigZagOffset)
anime({
  targets: zigZagPath,
  // animate from the full length to 0
  strokeDashoffset: [zigZagOffset, 0],
  duration: 3000,
  loop: true,
  direction: 'alternate',
  easing: 'easeInOutSine'
})

.conveyor {
  width: 200%;
}
```

Doing the same for our wave

```
// set up variables for our path
const wave = document.querySelector('#wave path')
const waveOffset = anime.setDashoffset(wave)
wave.setAttribute('stroke-dashoffset', waveOffset)
anime({
  targets: wave,
  strokeDashoffset: [waveOffset, 0],
  duration: 2000,
  loop: true,
  direction: 'alternate',
  easing: 'easeInOutSine'
})
```

Rotating crosses animation

```
anime({
  targets: '#crosses path',
  rotate: '1turn',
  delay: (el, i) => i * 100,
  duration: 1200,
  loop: true,
  direction: 'alternate',
  easing: 'easeInOutSine'
})
```

Duplicating our elements

```
// our duplicateHtml duplicates an elements content however many times we tell it
const duplicateHtml = (element, quantity) => {
  // this makes an array with a number of empty spaces
  const newContent = new Array(quantity)
    // fill the array with the content we want to duplicate
    .fill(element.innerHTML)
    // because it' s an array we want to join it all together into a big text string
    .join('')
  // then we replace that html inside of our element
  return element.innerHTML = newContent
}

// we can make 8 of them like this
const crosses = document.querySelector('#crosses')
duplicateHtml(crosses, 4)
```

Mini dots scattered animation

```
const dots = document.querySelectorAll('#dots .dot');
dots.forEach(dot => {
  anime({
    targets: dot,
    rotate: (el, i) => anime.random(90, 360),
    delay: (el, i) => anime.random(0, 500),
    duration: (el, i) => anime.random(250, 750),
    loop: true,
    easing: 'easeInOutSine',
    direction: 'alternate',
    autoplay: true
  });
});
```

Mini circles animation

```
anime({
  targets: '.circle',
  scale: [0, 1.2],
  delay: (el, i) => i * 100,
  duration: 250,
  loop: true,
  direction: 'alternate',
  easing: 'easeInOutSine'
})
```

Flashing rectangles animation

```
anime({
  targets: '.flash',
  // we put the colors into an array and grab each one by its index
  backgroundColor: (el, i) => ['#fff636', '#cb89fc', '#fc3035', '#77ebfd'][i],
  // apply a random delay to each one
  delay: (el, i) => anime.random(50, 100),
  duration: 500,
  loop: true,
  direction: 'alternate',
  easing: 'easeInOutSine'
})
```

Creating rectangles and squares

We can create rectangles in SVG by copying our SVG code from either Sketch or Illustrator and then modifying it. Or we can write out our `rect` elements by hand!

```
<svg width="200px" height="200px" viewBox="0 0 200 200" version="1.1"
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
 <desc>Created with Sketch.</desc>
 <defs></defs>
 <g stroke="#fff" stroke-width="10px" fill="none" fillRule="evenodd">
  <rect height="100%" width="100%" x="50%" y="50%" />
 </g>
</svg>
```

- `width` our rectangle width
- `height` our rectangle height

Rotating squares animation

```
anime({
 targets: '#squares rect',
 // we need to force the translate position to center the squares
 translateX: ['-50%', '-50%'],
 // we’ re not actually animating them, it’ s a bit of a hack
 translateY: ['-50%', '-50%'],
 // animate rotation from 45 to 0 to -45
 rotate: [45, 0, -45],
 // delay each one incrementally by 100ms
 delay: (el, i) => 100 * i,
 duration: (el, i) => 750,
 loop: true,
 easing: 'easeInOutSine',
 direction: 'alternate'
})
```