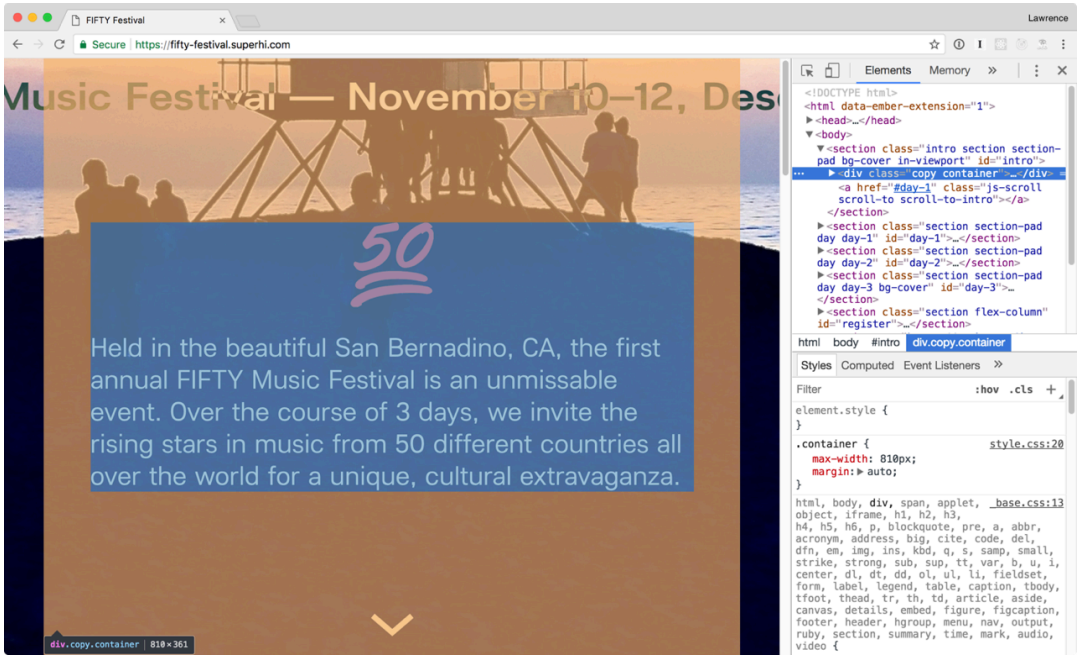
SuperHi-Advanced-Week1

Embedding font files in our CSS
在 CSS 中导入字体文件

```
/* we link to our web font files using @font-face */
@font-face {
  font-family: 'MyWebFont';
  src: url('myfont.woff2') format('woff2'),
  url('myfont.woff') format('woff'),
  url('myfont.ttf') format('truetype');
}

/* We can then use the font using its name… */
body {
  font-family: 'MyWebFont', sans-serif;
}
```



With display:flex; on our .section and margin:auto; on our .container we can center our element vertically and horizontally


```
/* add a mix-blend-mode to invert the text over the background */
.text {
  mix-blend-mode: difference;
}
```

Declaring animations using @keyframes

```
/* we declare what our styles should be at various steps */
@keyframes fadeIn {
  0% { opacity: 0; }
  100% { opacity: 1; }
}

/* we can include our animation on elements */
.container {
  animation: fadeIn 3s once;
}
```

Variables in modern Javascript

```
// rather than writing var, we can now put const
// it's a modern js feature (short for constant)
const name = 'lawrence'

// create an array with 8 empty spaces and then fill it with our `name` variable
const wholeLottaLawrence = new Array(5).fill(name)
→ ['lawrence', 'lawrence', 'lawrence', 'lawrence', 'lawrence']

// join each of the items in the array with the string ' and '
const joinedWithAnd = wholeLottaLawrence(' and ')
→ 'lawrence and lawrence and lawrence and lawrence and lawrence'
```

Using helper classes to do one thing

Something useful we can do in CSS is to create classes that do just one thing for us, and we can then attach them in multiple places in our HTML. The benefit here is we can reduce duplicate code and isolate common design patterns.

```
// here we set up a class to cover our background
.bg-cover {
  background-size: cover;
  background-repeat: no-repeat;
  background-position: center;
}
// center our text
.center {
  text-align: center;
}
```

Selecting elements with jQuery

```
// finds us all the divs on the page
$('div')
```

Selecting elements with vanilla Javascript

```
// finds us the first div on the page
document.querySelector('div')

// finds us all the divs on the page
document.querySelectorAll('div')
```

Changing content using jQuery

```
// finds us all the divs and change their content
$('div').html('jurassic park!')
```

Changing content with vanilla Javascript

```
// finds us the first div on the page and change its content
document.querySelector('div').innerHTML = 'jurassic park!'
```

Select items in a sequence

Using nth-of-type and nth-child selectors we can select elements that match a pattern in a sequence.

```
// select every 3rd .element in the series
.element:nth-of-type(3n) {
  background-color: yellow;
}
```

…some more examples:
https://css-tricks.com/useful-nth-child-recipies

Animating elements using Javascript

We can animate elements directly from Javascript using somewhat familiar code to what we're used to with CSS animations.

```
document.querySelector('.shape').animate([
  // keyframes
  { transform: 'translateY(0px)' },
  { transform: 'translateY(-300px)' }
], {
  // timing options
  duration: 1000,
  iterations: Infinity
})
```

A difference between jQuery and vanilla Javascript

When we grab elements using jQuery, it stores them using an array (list of things).

```
// gives us back an array of divs
$('div')
→ [div, div, div]
```

jQuery actually does a bit of magic for us, looping through all of the elements that it selects and performing actions on each individual one.

```
// jQuery will loop over all the individual divs
// it will then run .html('hello') on each one
$('div').html('hello')
```

In vanilla Javascript we need to loop over them
With vanilla Javascript and using querySelectorAll, we need to make sure we loop through all of our elements and perform whatever functionality we want on each individual one.

```
// selects all buttons, loops through each one and changes its content
document.querySelectorAll('button').forEach(button => {
  // here we can do something with each `button`
  button.innerHTML = 'hello'
})
```

What's the arrow thing about?

```
// in older javascript we'd write
['lawrence', 'rik'].forEach(function(person) {
  // do something with each person ('rik' and 'lawrence')
})

// in modern javascript we can write
['lawrence', 'rik'].forEach(person => {
  // do something with each person
})
```

Getting the index inside of a loop

Inside of a loop we can also get access to the index of each item in our array, which is really useful if we want to perform some functionality like adding a transition delay to each element in a series.

```
const team = ['lawrence', 'rik', 'milan', 'ryan', 'krista', 'adam']
// inside of forEach, we have the second index variable which increments each time
// it starts at 0 because arrays are zero-index (they start at zero)
team.forEach((person, index) => {
const nameAndIndex = person + ' is ' + index
  console.log(nameAndIndex)
})
→ ['lawrence is 0', 'rik is 1', 'milan is 2'] // and so on…
```

Joining strings together in modern Javascript

```
// in older javascript we would write
var name = 'lawrence'
var age = 26
var profile = name + ' is ' + age

// in modern javascript we can write
const name = 'lawrence'
const age = 26
const profile = `${name} is ${age}`
```

Finding elements only inside other elements

If we wanted to find .shape classes only that appear inside of our first div, we can do that using querySelector too!

```
// select our first div
const container = document.querySelector('div')

// select all .shapes inside of our container
const shapes = container.querySelectorAll('.shape')
```

Adding a delay to each element in a sequence

If we wanted to find .shape classes only that appear inside of our first div, we can do that using querySelector too!

```
// select all of our .shapes from the html
const circles = document.querySelectorAll('.circle')

// loop through each shape and set a transition delay using its index
circles.forEach((circle, index) => {
// each circle will get 0ms, 100ms, 200ms etc. set on it
  circle.style.transitionDelay = (index * 100) + 'ms'
})
```

Setting CSS properties in Javascript

Javascript likes to camelCase things. Camel case is just like how Apple name their products iPhone, iPad and iMac. The first word starts with lowercase and then every other word starts with uppercase.

```
// setting the background-color css property using javascript
const circle = document.querySelector('.circle')
circle.style.backgroundColor = 'green'

// they follow this pattern
`font-size` in CSS would be `fontSize` in Javascript
`line-height` would be `lineHeight`
```

Adding and removing classes

```
// select the first div and add a class of 'active' to it
document.querySelector('div').classList.add('active')

// select the first div and remove a class of 'active' to it
document.querySelector('div').classList.remove('active')
```

Scrolling to elements smoothly

We can scroll to elements on our page in a smooth fashion using Javascript's new scrollIntoView functionality.

```
// find the first div on our page and scroll to it using smooth behaviour
document.querySelector('div').scrollIntoView({
  behavior: 'smooth'
})
```

Performing actions on events in jQuery

In jQuery we can watch out for events using the .on() method. Under the surface jQuery will loop through each anchor tag and do the work of attaching the functions on click for us.

```
// the jquery version will grab all of the anchor tags
// it will console.log us a message each time we click one
$('a').on('click', () => {
  console.log('jackanacknory!')
})
```

Performing actions on events in Javascript

If we want to run a function every time we click on an anchor tag on our page, we have to loop through and tell Javascript to 'listen' for the particular event.

```
// here we grab all of the links and forEach through each one
// with each one we use link.addEventListener to watch our for clicks
document.querySelectorAll('a').forEach(link => {
  link.addEventListener('click', () => {
    console.log('jackanacknory!')
  })
})
```

Events in Javascript

Whenever we run functions triggered by events in Javascript, we get access to a whole host of data about that particular event. Almost like a snapshot of what happened, things like the mouse position, the scroll position, the time it happened and loads more. This is how things like Google Analytics work with understand user interaction and behavior.

```
document.querySelectorAll('a').forEach(link => {
  link.addEventListener('click', event => {
    // here we have access to `event` which contains all the snapshot data
  })
})
```

A bit about flexbox

By default display: flex; stacks all elements side by side and they will not wrap onto a new line.

Stacking items on top of each other

By using flex-direction: column; we can tell our boxes to stack on top of each other rather than side-by-side.

But that only puts one box per column?

flex-direction: column; is perfect when you only want one box stacking on top of another. But if you wanted say two boxes in your first row, and one in the second you could say:

```
// here we say to flexbox that it's okay to put boxes on new lines
// now they will will the available space, otherwise they'll sit on a new line
.boxes {
  display: flex;
  flex-wrap: wrap;
}
```

…more info on flexbox:
https://css-tricks.com/snippets/css/a-guide-to-flexbox