Run `git pull` from the public folder to follow along today.

This is the lecture I should have given before the D3 lecture.

JavaScript and Svelte

DSC 106: Data Visualization

Sam Lau

UC San Diego

Join at

slido.com

#3821

Announcements

Lab 7 (Scrollytelling) out, due tomorrow

Final Project proposal due tomorrow

FAQs:

1. When will Project 3 peer feedback be out? Within next few days. Feedback will be due 3 (business) days after emails are sent.
2. When will Project 2 peer feedback be returned to us? Within next few days.

What was your biggest struggle with implementing Project 3?

Join at
slido.com
#3821

To get participation for rest of lecture, submit one question on Slido at any point.

Join at
slido.com
#3821

pandas code executes from top to bottom

Selecting columns

Selecting columns in `babypandas` 👶🐼

- In `babypandas`, you selected columns using the `.get` method.
- `.get` also works in `pandas`, but it is not **idiomatic** – people don't usually use it.

In [26]:
```
dogs
```

Out[26]:

breed	kind	lifetime_cost	longevity	size	weight	height
Brittany	sporting	22589.0	12.92	medium	35.0	19.0
Cairn Terrier	terrier	21992.0	13.84	small	14.0	10.0
English Cocker Spaniel	sporting	18993.0	11.66	medium	30.0	16.0
...	...	...	...	...	...	...
Bullmastiff	working	13936.0	7.57	large	115.0	25.5
Mastiff	working	13581.0	6.50	large	175.0	30.0
Saint Bernard	working	20022.0	7.78	large	155.0	26.5

43 rows × 6 columns

In [27]:
```
dogs.get('size')
```

Out[27]:
```
breed
Brittany                 medium
Cairn Terrier             small
English Cocker Spaniel    medium
                           ...
Bullmastiff               large
Mastiff                   large
Saint Bernard             large
Name: size, Length: 43, dtype: object
```

In [28]:
```
# This doesn't error, but sometimes we'd like it to.
dogs.get('size oops!')
```

Svelte JS code runs once from top-to-bottom...

Selecting columns in `babypandas` 👶🐼

- In `babypandas`, you selected columns using the `.get` method.
- `.get` also works in `pandas`, but it is not **idiomatic** – people don't usually use it.

In [26]:
```
dogs
```

Out[26]:

breed	kind	lifetime_cost	longevity	size	weight	height
Brittany	sporting	22589.0	12.92	medium	35.0	19.0
Cairn Terrier	terrier	21992.0	13.84	small	14.0	10.0
English Cocker Spaniel	sporting	18993.0	11.66	medium	30.0	16.0
...	...	...	...	...	...	...
Bullmastiff	working	13936.0	7.57	large	115.0	25.5
Mastiff	working	13581.0	6.50	large	175.0	30.0
Saint Bernard	working	20022.0	7.78	large	155.0	26.5

43 rows × 6 columns

But sometimes snippets in the middle get re-run, how does that work??

In [27]:
```
dogs.get('size')
```

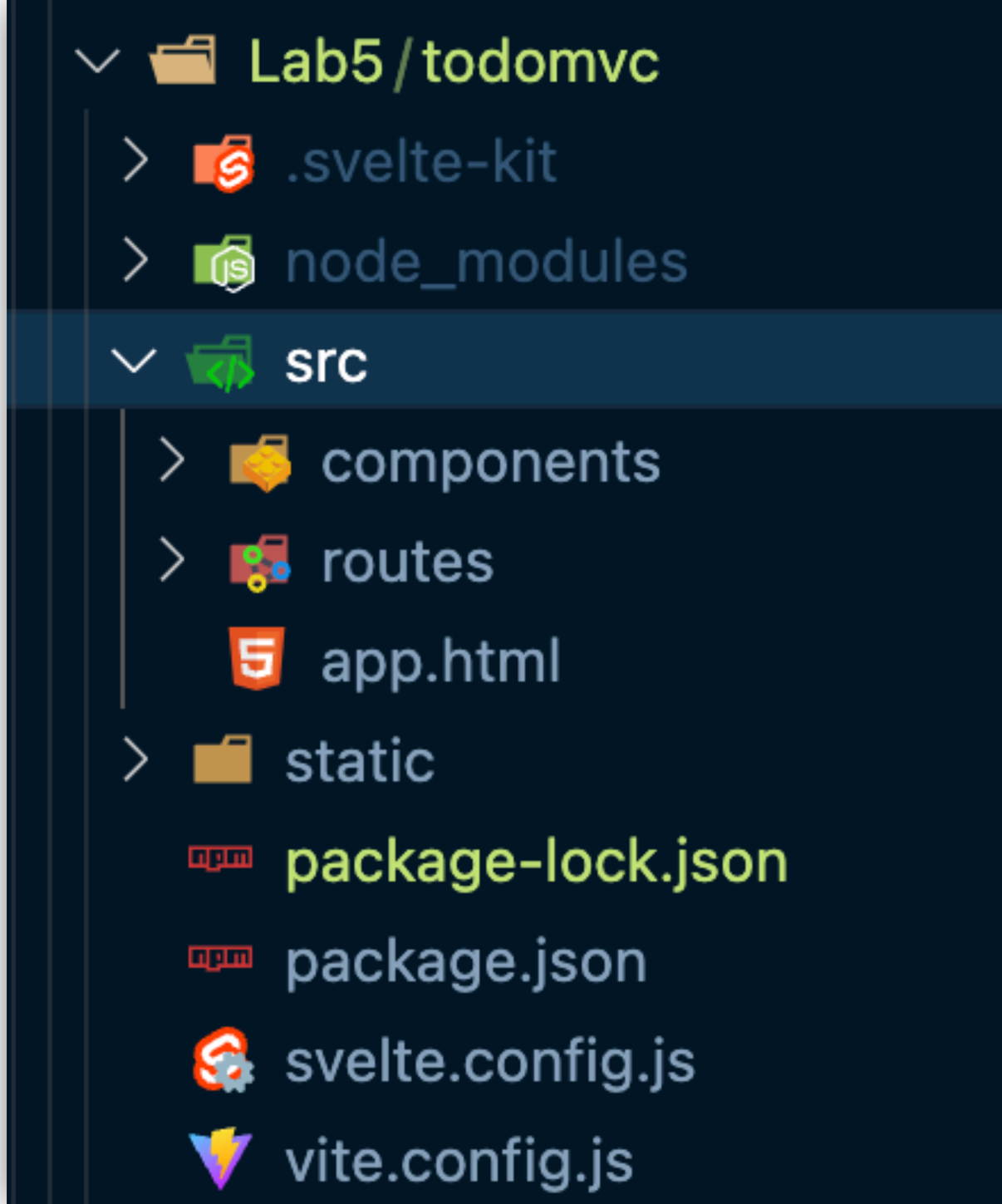
Out[27]:
```
breed
Brittany
Cairn Terrier
English Cocker Sp

Bullmastiff
Mastiff
Saint Bernard
Name: size, Length: 43, dtype: object
```

In [28]:
```
# This doesn't error, but sometimes we'd like it to.
dogs.get('size oops!')
```

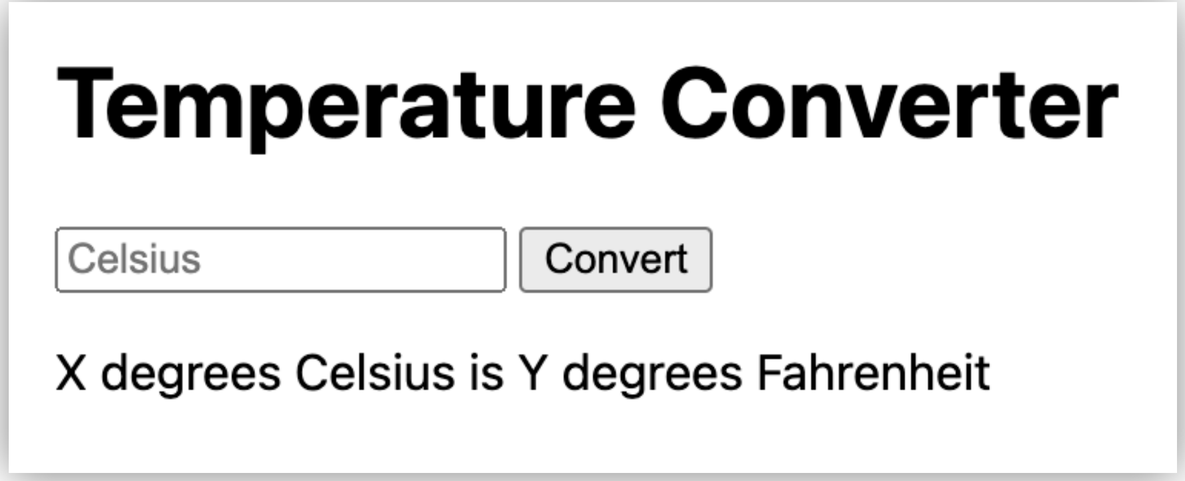
```svelte
<script>
  import * as d3 from 'd3';

  export let data;

  const width = 928;
  const height = 600;
  const marginTop = 20;
  const marginRight = 30;
  const marginBottom = 30;
  const marginLeft = 40;

  let svg;

  // Placeholders for the axis elements.
  let gx;
  let gy;

  $: x = d3
    .scaleUtc()
    .domain(d3.extent(data, (d) => d.date))
    .range([marginLeft, width - marginRight]);

  $: y = d3
    .scaleLinear()
    .domain(d3.extent(data, (d) => d.value))
    .nice()
    .range([height - marginBottom, marginTop]);

  $: max = d3.max(data, (d) => Math.abs(d.value));
```

Also, what do these files + folders mean?

And why do things break when I don't put them in the right folder?



Doing things the hard way: Plain JavaScript



Pseudocode:

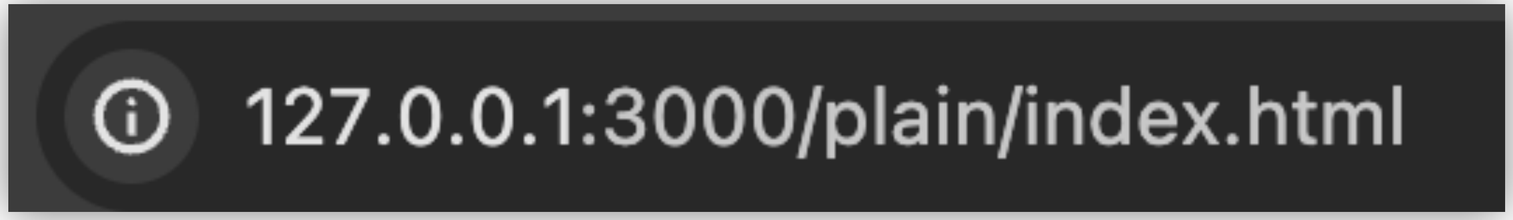
1. When we click "Convert", read value in Celsius box.
2. Convert value to F
3. Update text below box

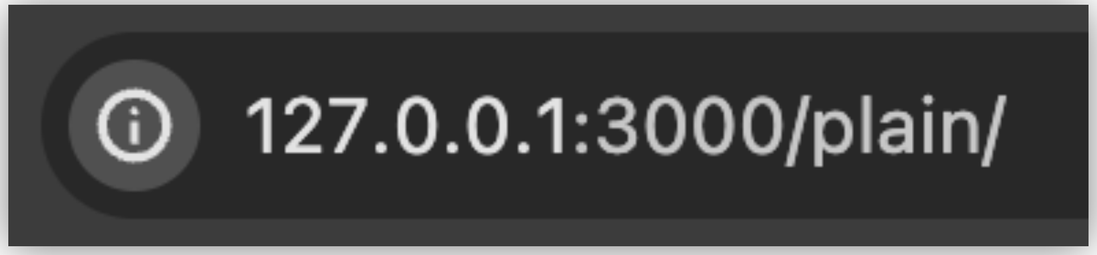
(demo)

JS approach:

1. Attach event listener to Convert button. Event handler reads value from Celsius box.
2. Convert value to F
3. Replace text of the element below.

Typing a URL into address bar only asks for one HTML file (index.html in this case):





index.html is appended if it isn't in URL, so this is the same.

Download, then render index.html

Download, then render index.html

HTML is also "executed" top-to-bottom:

```html
<html>
  <head>
    <title>Temperature Converter</title>

    <link rel="stylesheet" href="main.css" />
    <script src="main.js"></script>
  </head>
  <body>
    <h1>Temperature Converter</h1>

    <div id="converter">
      <input type="text" id="celsius" placeholder="Celsius" />
      <button id="submit" type="submit">Convert</button>
    </div>

    <div id="result">X degrees Celsius is Y degrees Fahrenheit</div>
  </body>
</html>
```

Download and run the main.css file

Download and run the main.js file

Render HTML to screen

Download, then render index.html

HTML is also "executed" top-to-bottom:

```
<html>
  <head>
    <title>Temperature Converter</title>

    <link rel="stylesheet" href="main.css" />
    <script src="main.js"></script>
  </head>
  <body>
    </div>

    <div id="result">X degrees Celsius is Y degrees Fahrenheit</div>
  </body>
</html>
```

Download and run the main.css file

Download and run the main.js file

What if these files are really big, or JS has an infinite loop??

Browser waits!

Download, then render index.html

HTML is also "executed" top-to-bottom:

```html
<html>
  <head>
    <title>Temperature Converter</title>

    <link rel="stylesheet" href="main.css" />
    <script src="main.js"></script>
  </head>
  <body>
    <h1>Temperature Converter</h1>

    <div id="converter">
      <input type="text" id="celsius" placeholder="Celsius" />
      <button id="submit" type="submit">Convert</button>
    </div>

    <div id="result">X degrees Celsius is Y degrees Fahrenheit</div>
  </body>
</html>
```

Download and run the main.css file

Download and run the main.js file

Render HTML to screen

`js-lecture/plain01/`

(demo)

```html
<html>
  <head>
    <title>Temperature Converter</title>

    <link rel="stylesheet" href="main.css" />
    <script src="main.js"></script>
  </head>
  <body>
    <h1>Temperature Converter</h1>

    <div id="converter">
      <input type="text" id="celsius"
      <button id="submit" type="submi
    </div>

    <div id="result">X degrees Celsius is Y degrees Fahrenheit</div>
  </body>
</html>
```

What happened?

```js
const button = document.getElementById('submit');

button.addEventListener('click', (event) => {
  event.preventDefault();
  console.log(event);
});
```

Runs before rest of HTML loads. There are no HTML elements in document!

`js-lecture/plain02/`

(demo)

Let's walk through the code line by line

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault
    const celsiusTag = d
    const celsius = cels
    const fah = (celsius

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

When everything else is done loading, call this function.

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDef
    const celsiusTag                                    );
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

Now we have an HTML element.

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

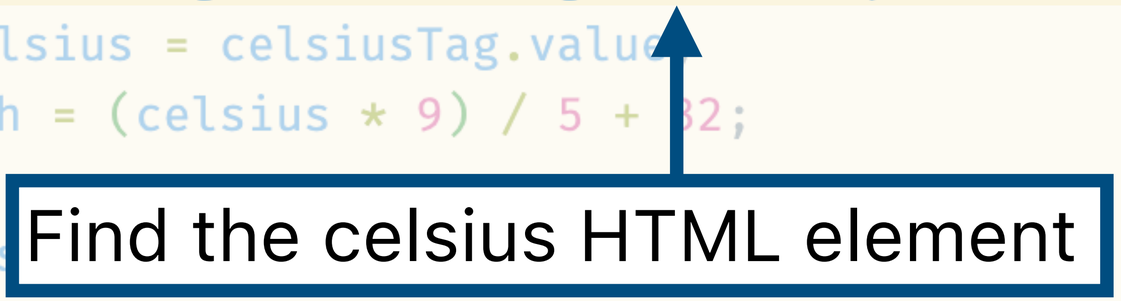
When button is clicked, called this function

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = celsius * 9 / 5 + 32;
    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

Stop the default behavior of the button.

Doesn't do anything here, but we usually want this when we're working in JS.

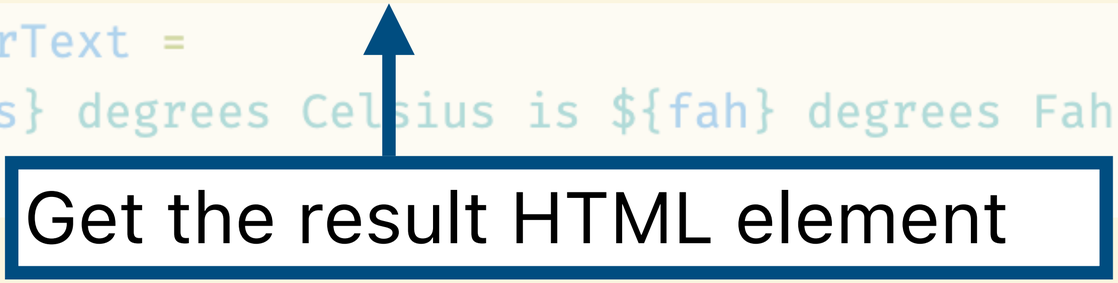
```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value
    const fah = (celsius * 9) / 5 + 32;

    const res                                    ;
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```



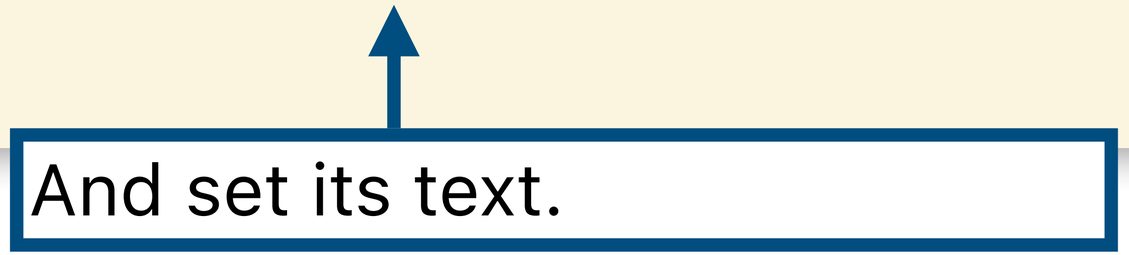
```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} d                    heit.`;
  });
});
```

Get its value, then convert to F

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```



```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```



```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fah
  });
});
```

JS code always runs top-to-bottom, but we use event handlers to **delay execution**...

And to **rerun** code in response to user input.

```
document.addEventListener('DOMContentLoaded', () => {
  const but

  button.ad
    event.p
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

This event only fires once (once the page is done loading), so this function only called once.

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    ev
    co                                          elsius');
    co
    co

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

This event fires every time a user clicks the button, so this function can get called many times.

Why Svelte?

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

How many times do we need to search through the entire document for an ID?

Once in the beginning...

How many times do we need to search through the entire document for an ID?

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

And potentially many times in this event handler

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    const fah = (celsius * 9) / 5 + 32;
  });
});
```

Core logic of this function is really just one line. Rest is dealing with HTML.

Svelte intuition: Let functions focus on data, and have HTML update automatically.

Plain JS

```
document.addEventListener('DOMContentLoade
  const button = document.getElementById('submit');

  button.addEventListener
    event
    const
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById
    result.innerText =
      `${celsius} degrees Celsius is ${fah
  });
});
```

2 lines of JS

JS just manipulates data

No repetition in HTML

Svelte

```
<script>
  let celsius = 0;
  $: fah = (celsius * 9) / 5 + 32;
</script>

<main>
  <h1>Temperature Converter</h1>

  <div id="converter">
    <input bind:value={celsius} type="text" />
  </div>

  <div id="result">
    {celsius} degrees Celsius is
    {fah} degrees Fahrenheit
  </div>
</main>
```

Understanding Svelte Code

App.svelte

```
<script>…
</script>

<main>…
</main>

<style>…
</style>
```

JavaScript logic

HTML

CSS

One file instead of three!

`js-lecture/svelte/components/App.svelte`

(demo)

```
document.addEventListener('DOMContentLoaded', () => {
  const button =

  button.addEver
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

Svelte renders elements after DOM has loaded, so this event handler never gets called.

```
document.addEventListener('DOMContentLoaded', () => {
```

```
import { onMount } from 'svelte';
onMount(() => {
```

Replace with onMount if you want to write plain JavaScript and don't care about Svelte.

Warning: mixing onMount with Svelte reactivity ($:) can result in weird bugs!

```
    const result = document.getElementById('result');
    result
      `${
  });
});
```

js-lecture/svelte/components/App01.svelte

(demo)

```
import { onMount } from 'svelte';
onMount(() => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    const celsiusTag = document.getEle
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

Start by binding elements instead searching entire DOM.

Then, use Svelte events instead of addEventListener.

`js-lecture/svelte/components/App02.svelte`

(demo)

```svelte
<script>
  let celsiusTag;
  let button;
  let result;

  function updateFah() {
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  }
</script>
```

We can bind the value of the tag directly instead of needing to read it in.

We can bind values into HTML to update them automatically.

`js-lecture/svelte/components/App03.svelte`

(demo)

Before:

```
document.addEventListener('DOMContentLoaded', () => {
  const button = document.getElementById('submit');

  button.addEventListener('click', (event) => {
    event.preventDefault();
    const celsiusTag = document.getElementById('celsius');
    const celsius = celsiusTag.value;
    const fah = (celsius * 9) / 5 + 32;

    const result = document.getElementById('result');
    result.innerText =
      `${celsius} degrees Celsius is ${fah} degrees Fahrenheit.`;
  });
});
```

Event listener

HTML

Event listener

Event listener

HTML

HTML

Logic

HTML

HTML

```
<script>
  let celsius = 0;
  $: fah = (celsius * 9) / 5 + 32;
</script>

<main>
  <h1>Temperature Converter</h1>

  <div id="converter">
    <input bind:value={celsius} type="text" />
  </div>

  <div id="result">
    {celsius} degrees Celsius is
    {fah} degrees Fahrenheit
  </div>
</main>
```

Logic

Logic

Binding

Binding

```svelte
<script>
  let celsius = 0;
  $: fah = (celsius * 9) / 5 + 32;
</script>

<main>
  <h1>Temperature Converter</h1>

  <div id="converter">
    <input bind:value={celsius} type="text" />
  </div>

  <div id="result">
    {celsius} degrees Celsius is
    {fah} degrees Fahrenheit
  </div>
</main>
```

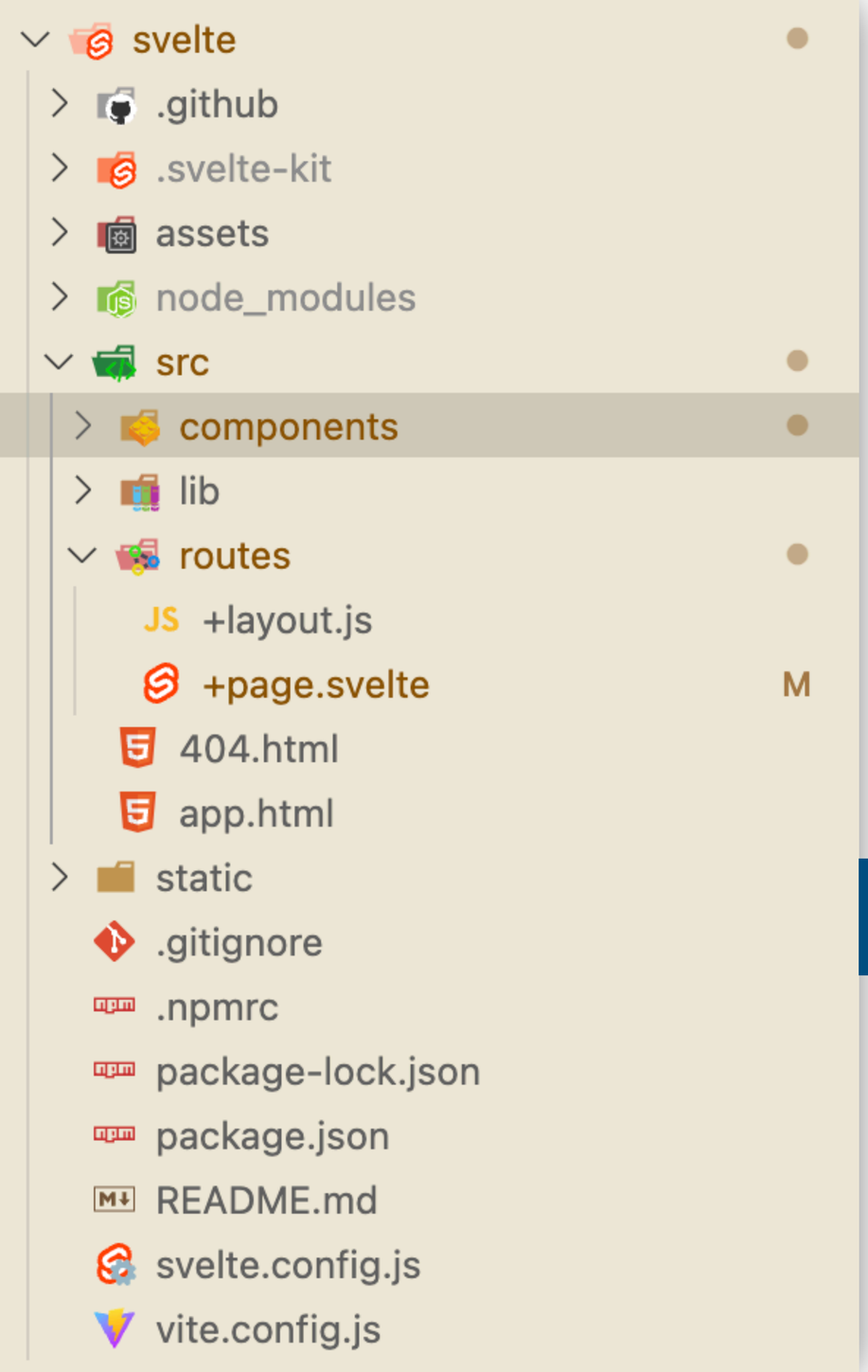
2. celsius automatically updates.

3. fah depends on celsius, so this line automatically runs since we used $:

1. When user types in this input...

4. The HTML depends on celsius and fah, so it updates too.

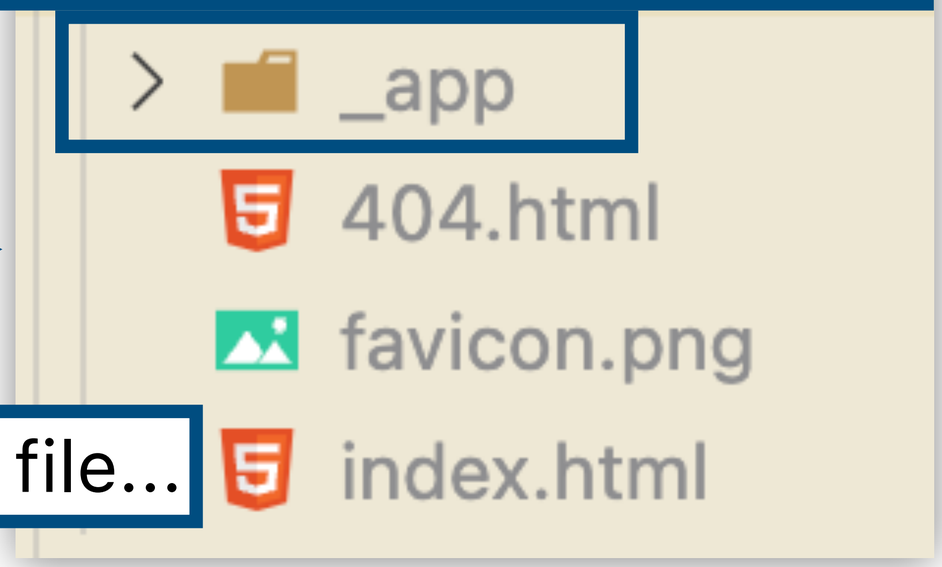
Understanding the Svelte Folder Structure



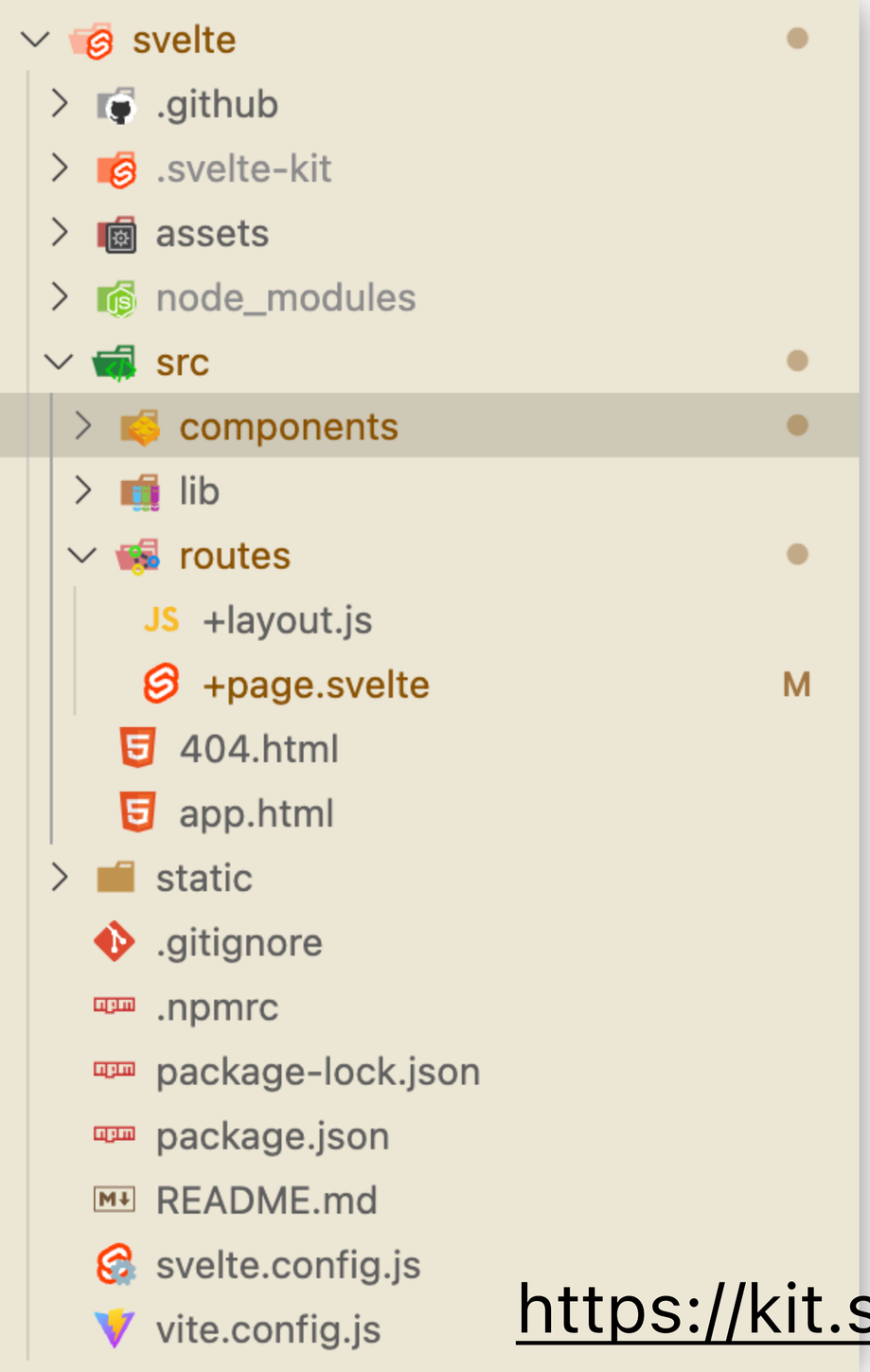
Svelte converts .svelte files into plain JS files. This is called **transpilation**.

...that loads CSS and JS files from here.

npm run build



Just one index.html file...



`.github/` has the config to auto-deploy GitHub pages

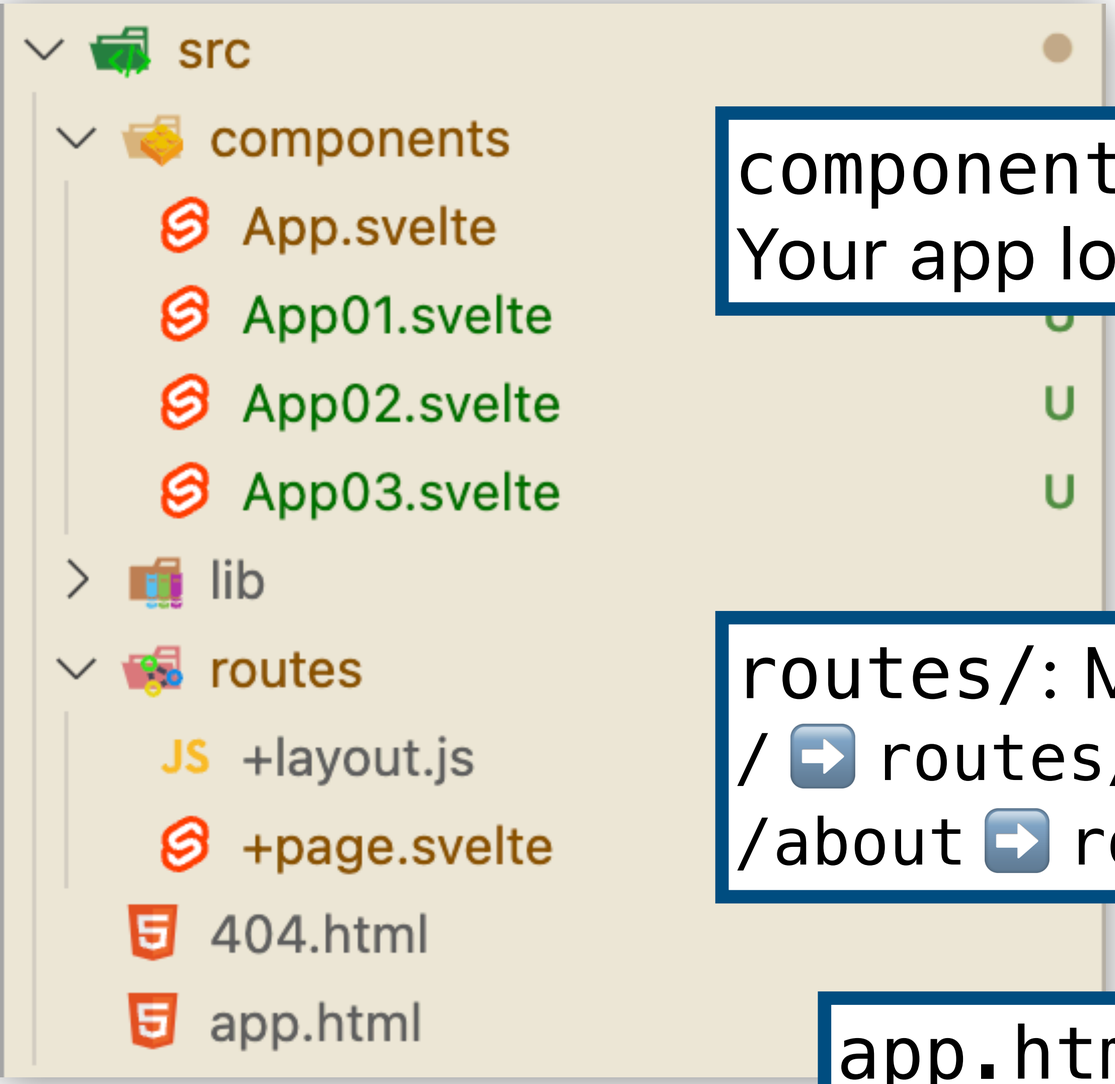
`node_modules/` has JS libraries (e.g. d3)

`src/` has original (source) .svelte files

`static/` contains files (e.g. data) that should just be copied (not transpiled) into the build

`package.json` lists the packages your project needs

https://kit.svelte.dev/docs/project-structure



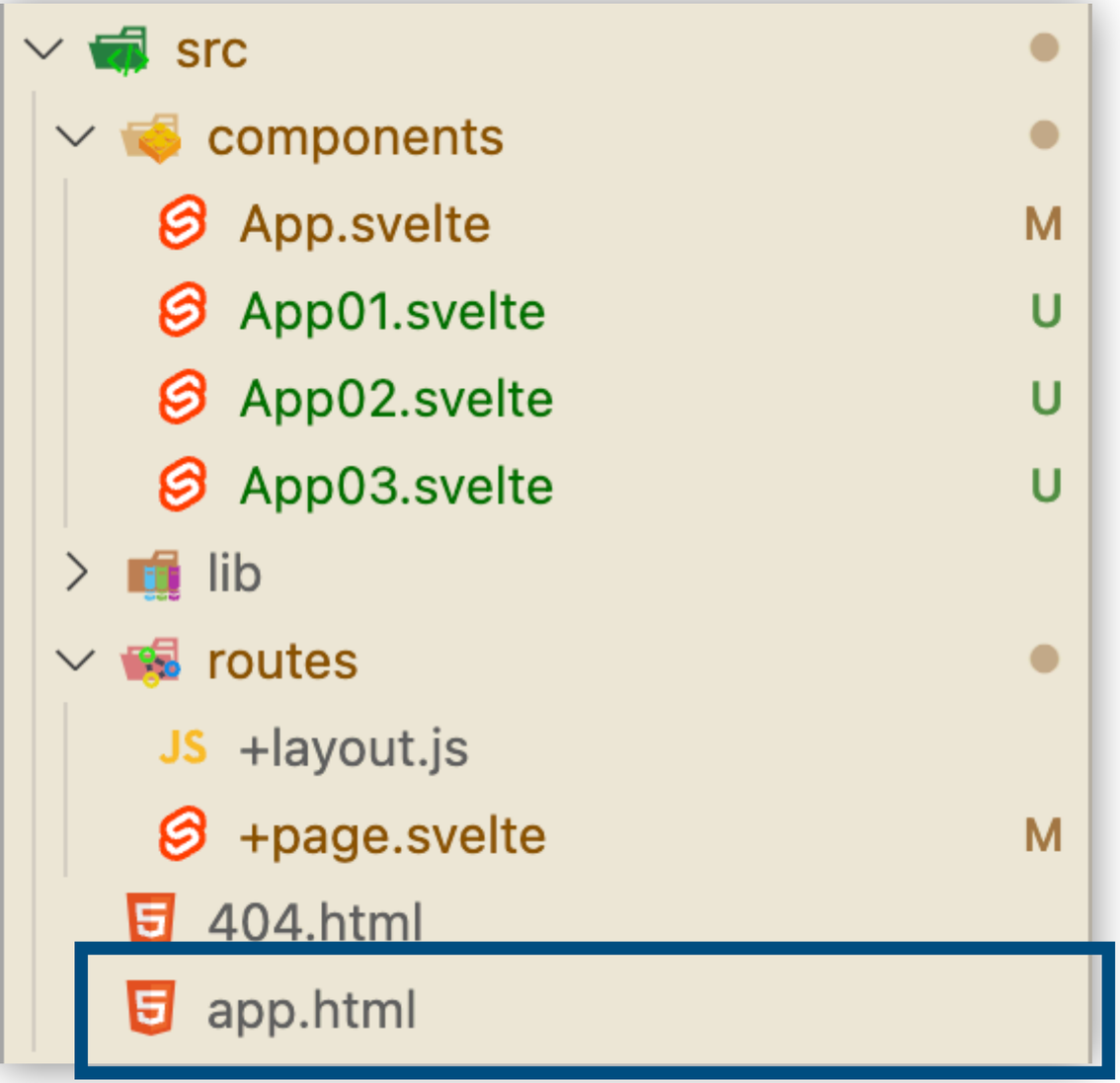
`components/`: Svelte components of your app. Your app logic will go here.

`routes/`: Maps URLs to Svelte files.
`/` ➡️ `routes/+page.svelte`
`/about` ➡️ `routes/about/+page.svelte`

`app.html`: Top-level HTML template

https://kit.svelte.dev/docs/project-structure

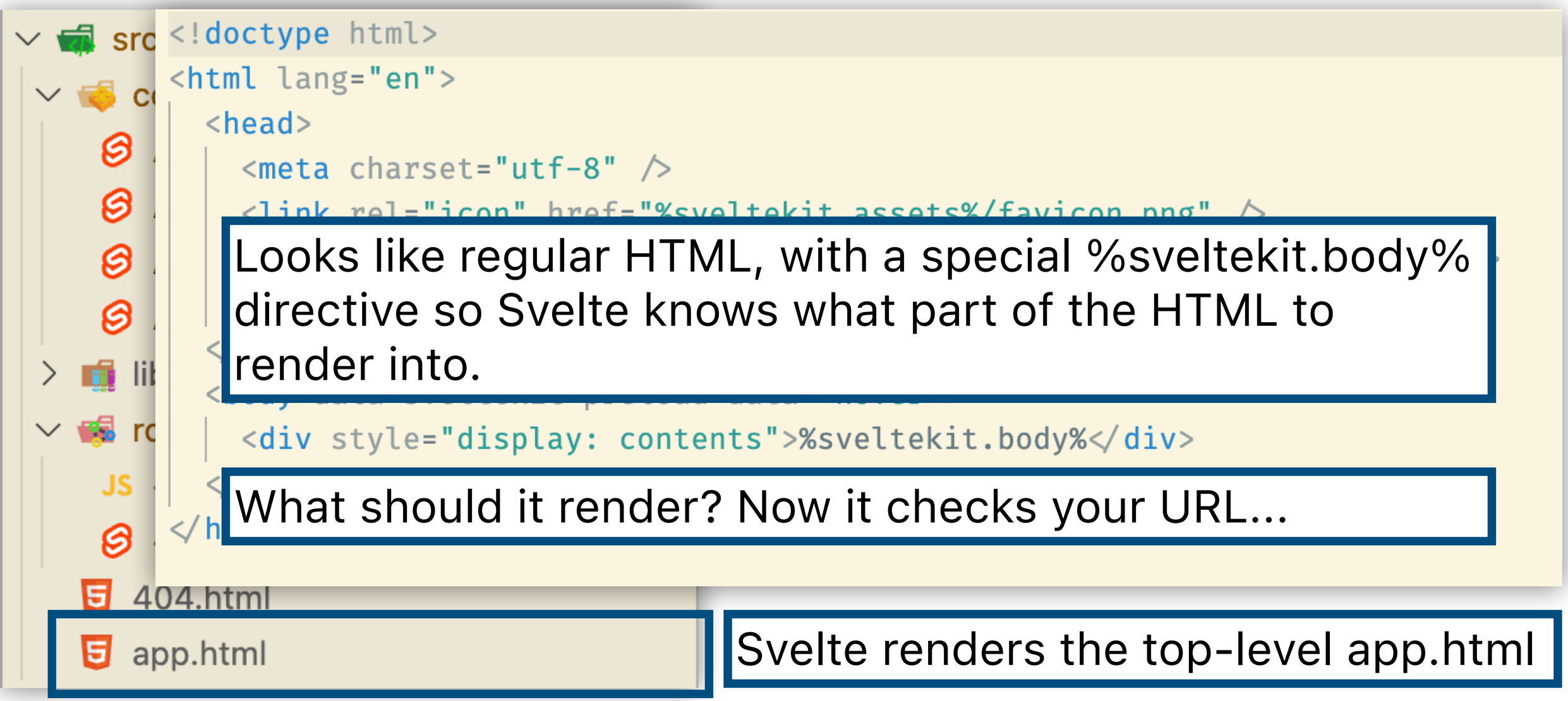
One mental model for how things work...



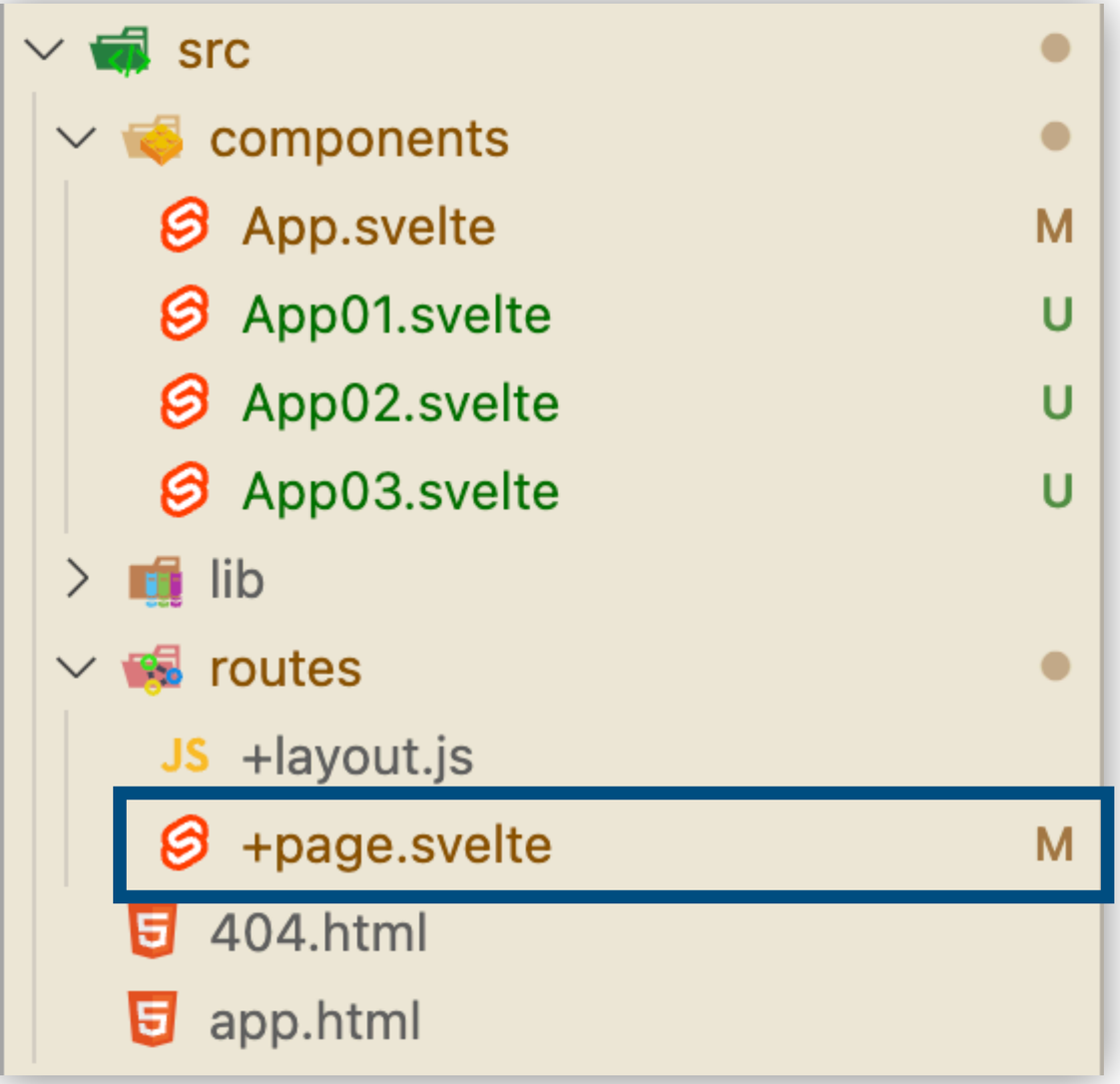
When browser visits your {id}.github.io/{repo}/ URL...

Svelte renders the top-level app.html

One mental model for how things work...



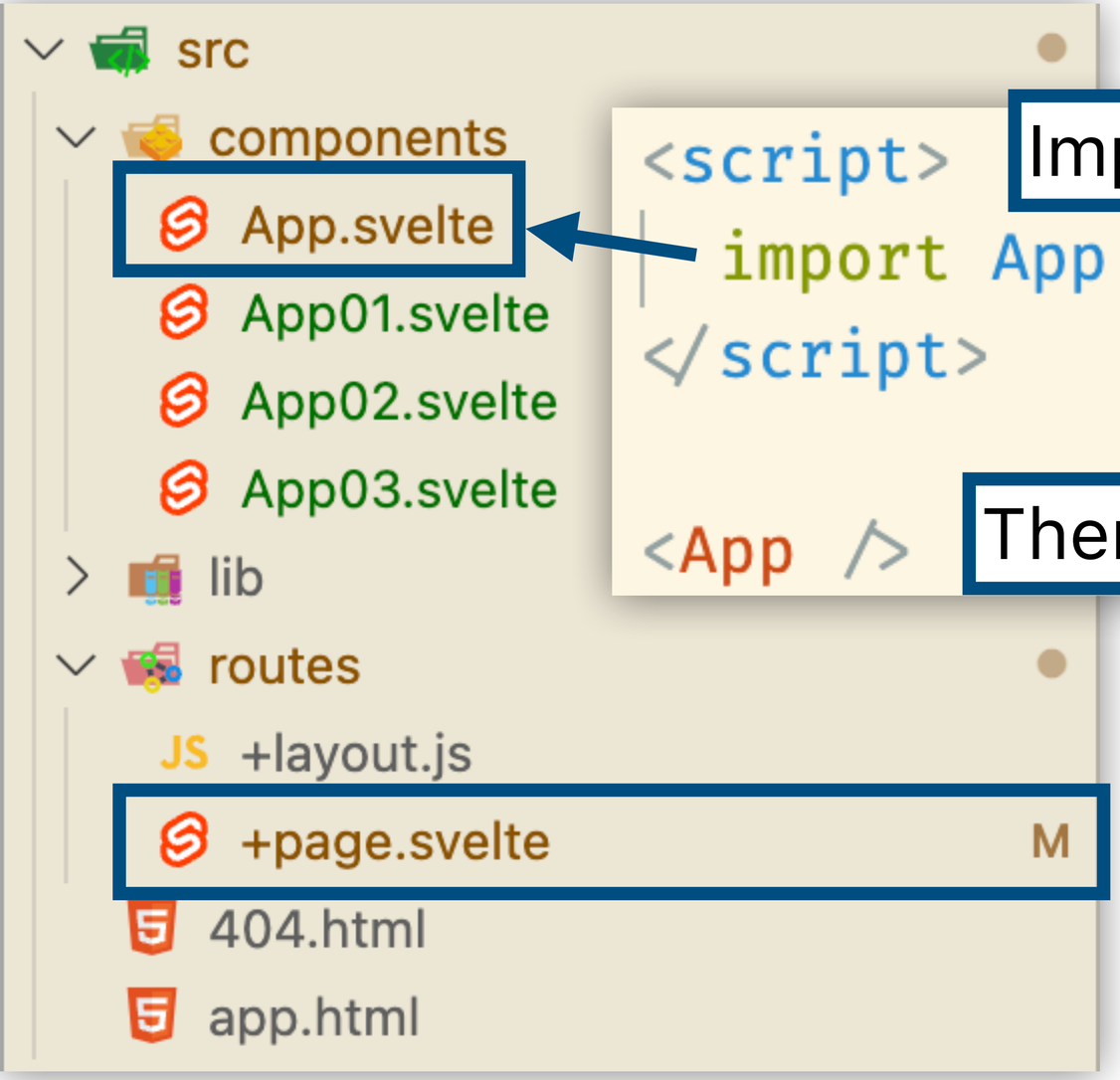
One mental model for how things work...



When browser visits your {id}.github.io/{repo}/ URL...

URL is the root of the website, so it looks for `routes/+page.svelte`

One mental model for how things work...



When browser visits your

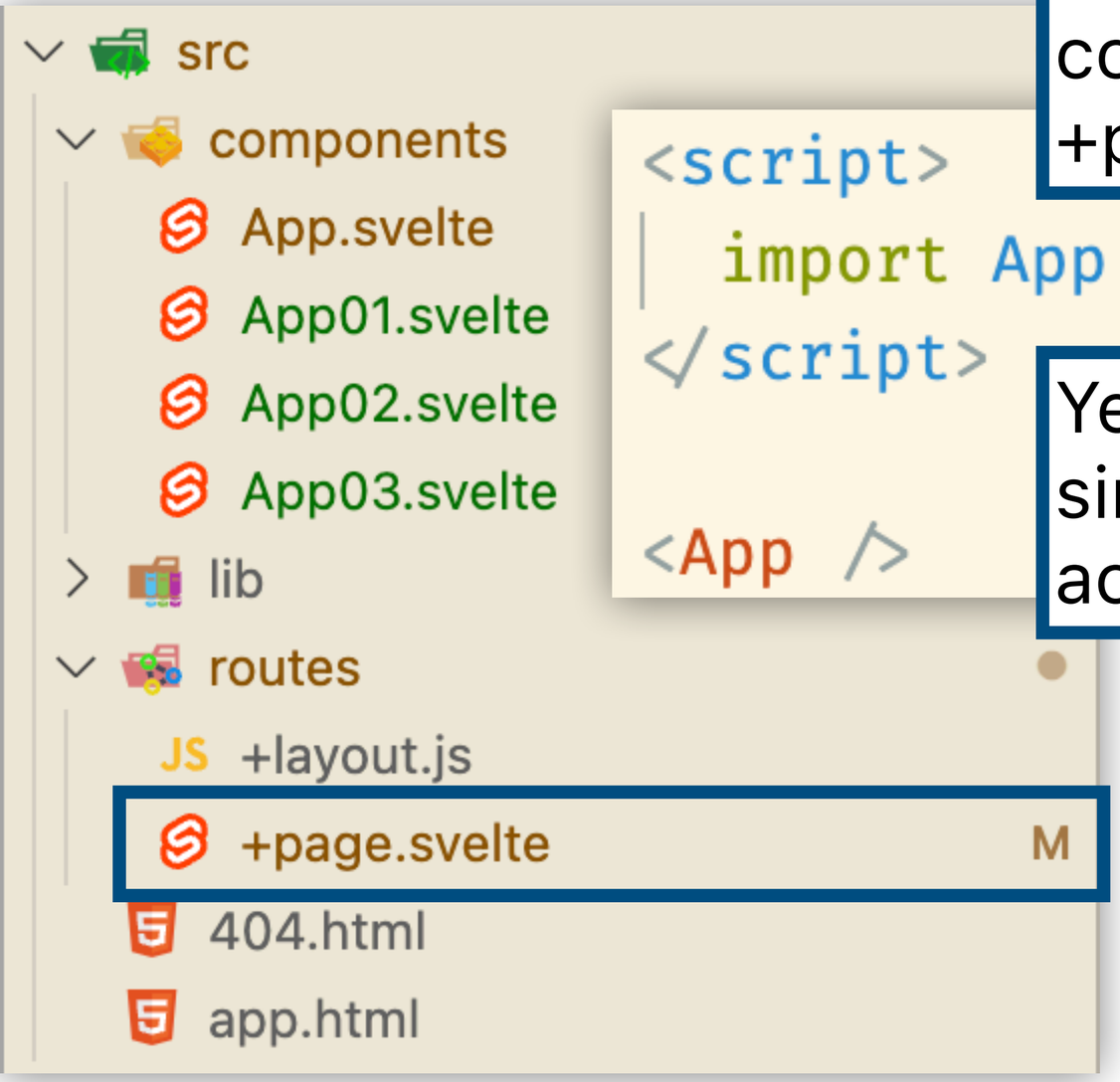
Import a component...

```
<script>
  import App from '../components/App.svelte';
</script>

<App />
```

Then render it into the page.

URL is the root of the website, so it looks for `routes/+page.svelte`

One mental model for how things work




```
<script>
  import App from '../components/App.svelte';
</script>

<App />
```

Doesn't that mean I could put all of my code from App.svelte into +page.svelte?

Yes, although it's not best practice, since components can get reused across pages of a web app.

URL is the root of the website, so it looks for `routes/+page.svelte`

Example: Name Grapher

`js-lecture/name-grapher/components/NameGrapher01.svelte`

(demo)

Example: Adding filtering to Name Grapher

`js-lecture/name-grapher/components/NameGrapher02.svelte`

(demo)

Example: Adding a tooltip to Name Grapher

`js-lecture/name-grapher/components/NameGrapher03.svelte`

(demo)