Svelte 101

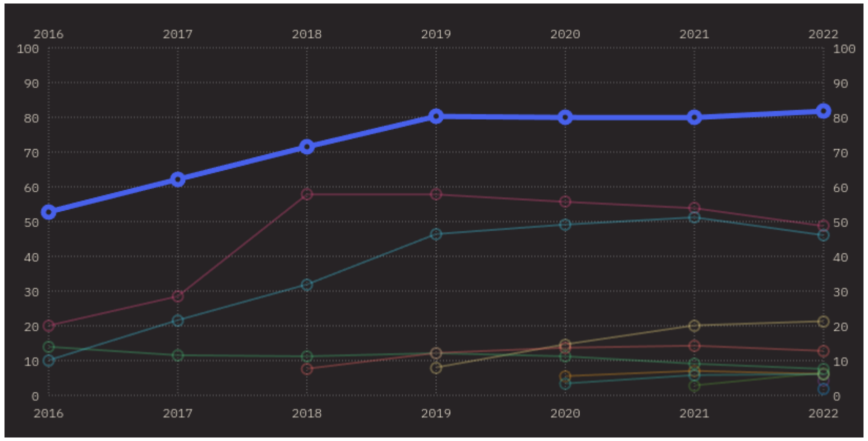
inovex

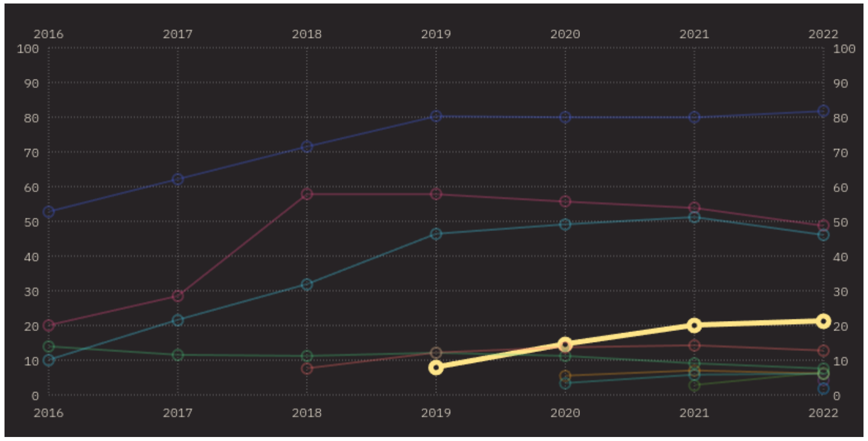
Manuel Ernst

- Software Engineer at inovex Erlangen
- Web-Development for over 20 years

www.linkedin.com/in/manuel--ernst

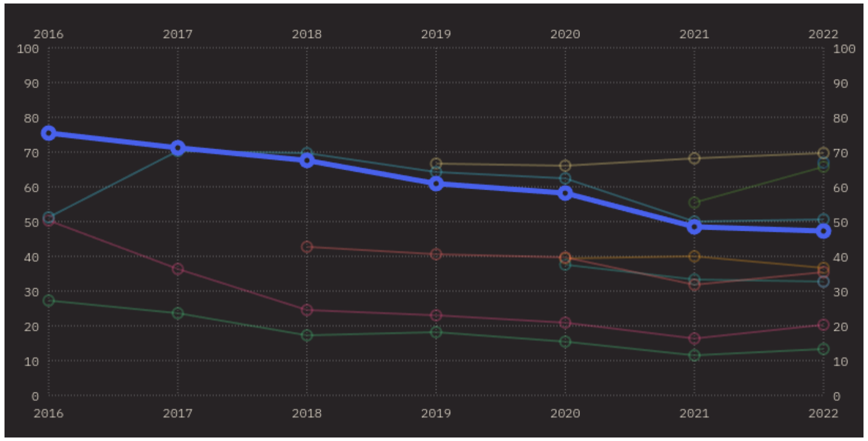
Usage

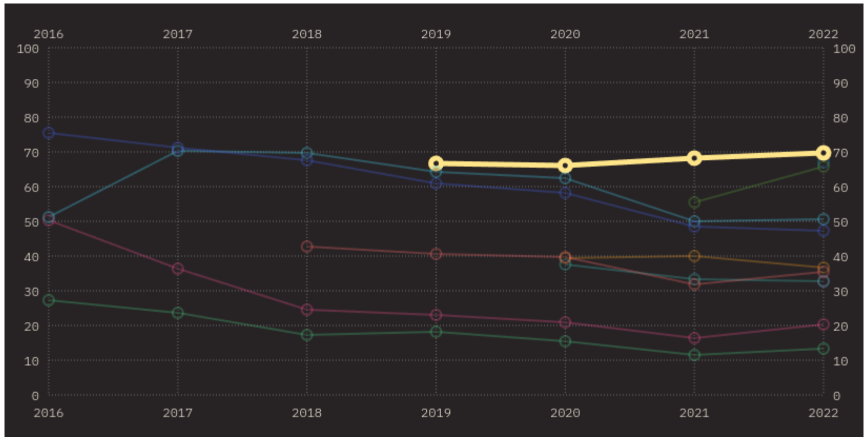




Source: State Of JS 2022

Interest





Source: *State Of JS 2022*

Popular Svelte Users

1Password

CNN

DB BAHN

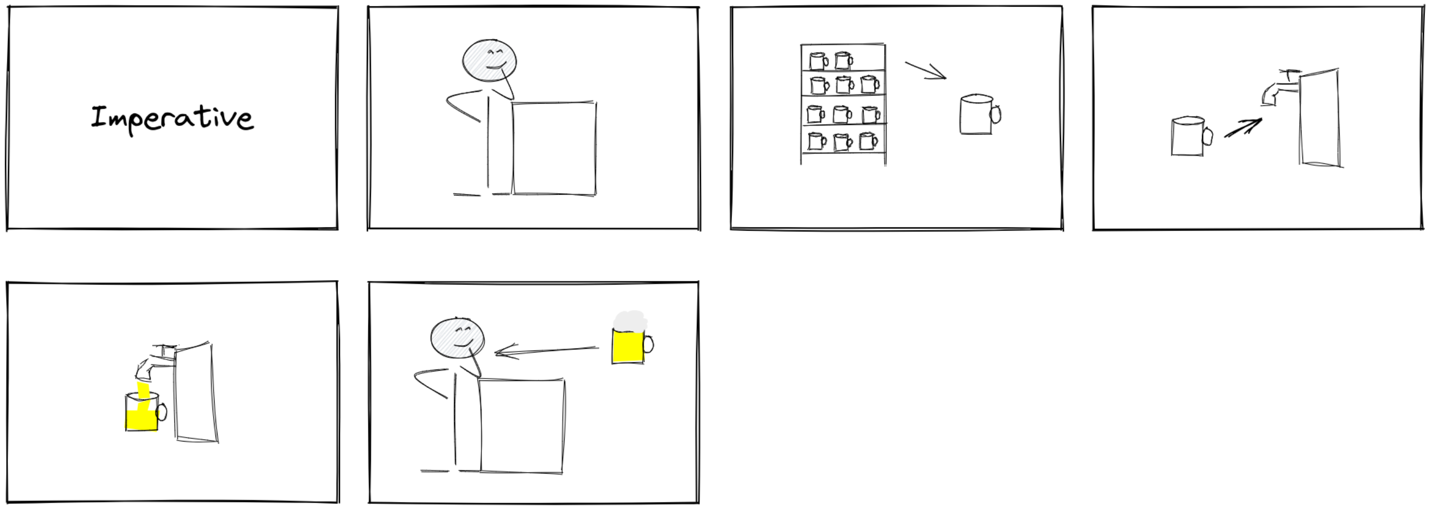
DECATHLON

IKEA®

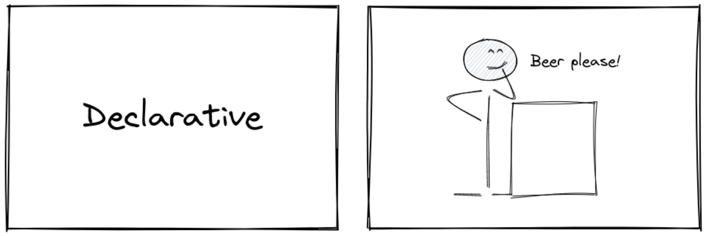
The New York Times

Spotify

Imperative vs. Declarative



Imperative vs. Declarative



DOM Intro

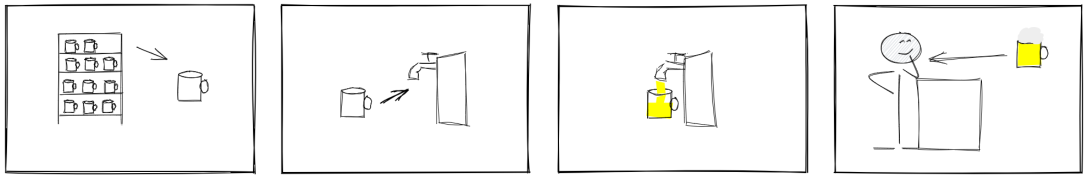
The DOM (**D**ocument **O**bject **Mo**del) is an *imperative* programming interface that is used to interact with the page markup.

It can be...

- inspected,
- amended,
- mutated.

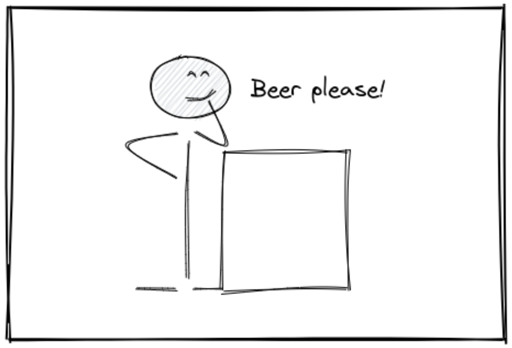
Imperative UI Definition

```javascript
let counter = 0

const button = document.createElement('button')
button.innerText = `clickcount: ${counter}`

button.addEventListener('click', () => {
  counter++
  button.innerText = `clickcount: ${counter}`
})

body.appendChild(button)
```

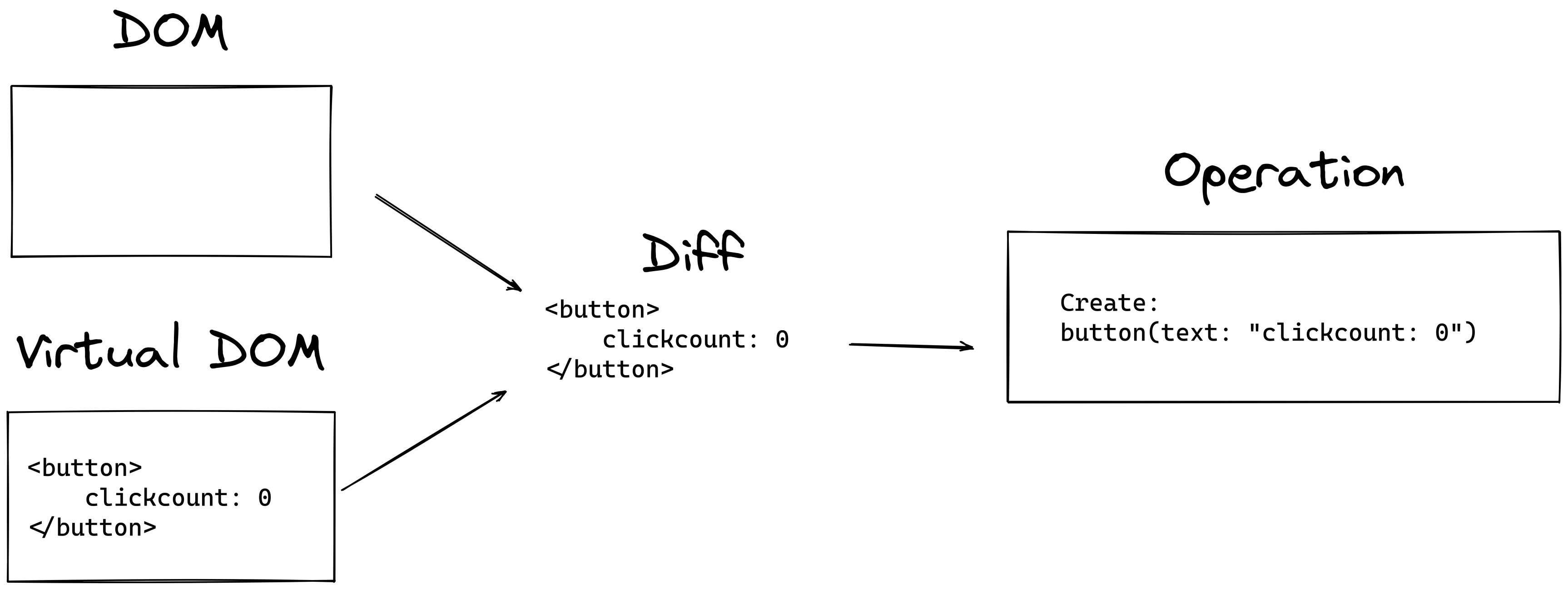


Declarative UI Definition

```
function Counter() {
  const [count, setCount] = useState(0)

  return (
    <button onClick={() => setCount(count + 1)}>
      clickcount: {count}
    </button>
  )
}
```



Runtime DOM Diffing

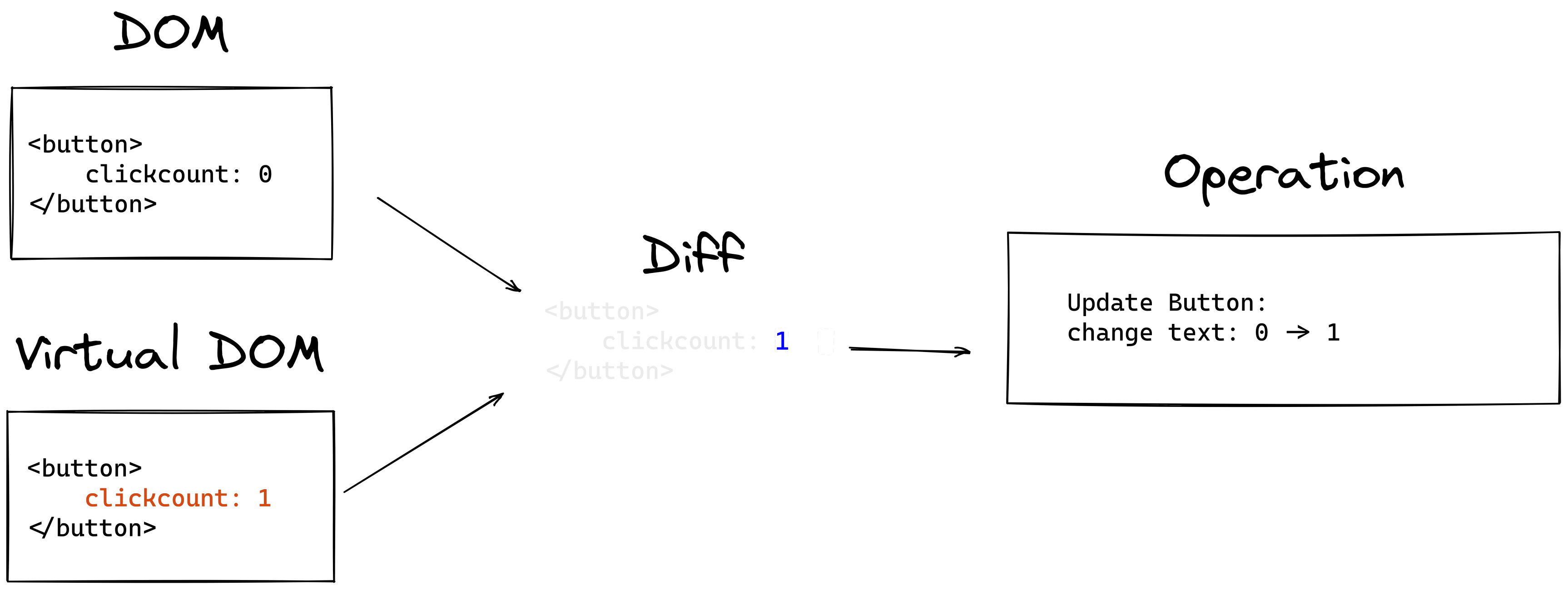


```
<button>
    clickcount: 0
</button>
```

```
<button>
    clickcount: 0
</button>
```

```
Create:
button(text: "clickcount: 0")
```

Runtime DOM Diffing



Feature Comparison

Rendering	Runtime	Buildtime
Runtime Environment	full-blown	minimal
State Management	BYO	integrated
Reactivity	-	integrated
Styling	BYO	integrated
Animations / Transitions	BYO	integrated

Svelte - Components

A component is a reusable self-contained block of code that encapsulates:

- Markup
- Styling
- Logic

Svelte - Components

```
<h2>Hello world!</h2>
```

Output

Hello world!

Templating

```
<script lang="ts">
  const name = 'world'
</script>

<h2>Hello {name}!</h2>
```

Hello world!

Output

Dynamic Attributes

```
<script lang="ts">
  const src = '/tutorial/image.gif'
</script>

<img src={src}>
```

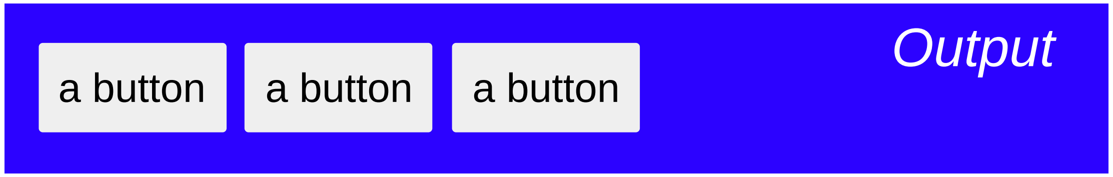
Output

```
<!-- short version -->
<img {src}>
```

Referencing Components

```
<!-- MyButton.svelte -->
<button>a button</button>
```

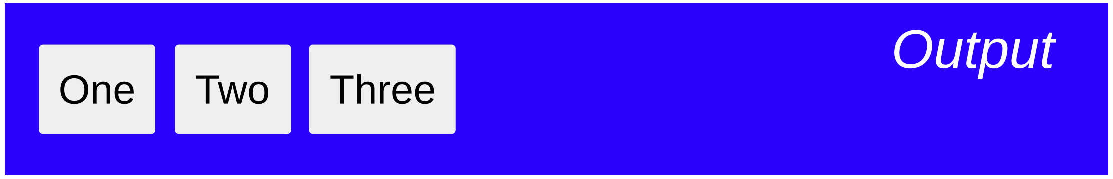
```
<!-- ButtonContainer.svelte -->
<script>
  import MyButton from './MyButton.svelte'
</script>

<MyButton />
<MyButton />
<MyButton />
```



Props for Components

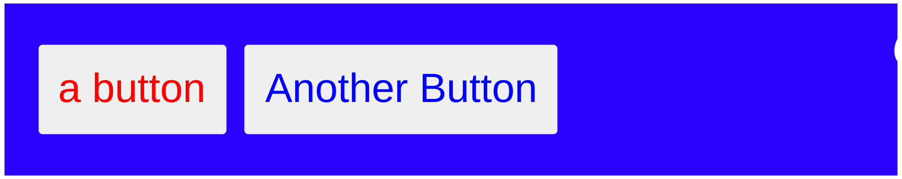
```
<!-- MyButton.svelte -->
<script>
  export let caption = 'a button'
</script>

<button>{caption}</button>
```

```
<!-- ButtonContainer.svelte -->
<script>
  import MyButton from './MyButton.svelte'
</script>

<MyButton caption="One" />
<MyButton caption="Two" />
<MyButton caption="Three" />
```



Styling

```
<!-- MyButton.svelte -->
<button>a button</button>

<style>
  button {
    color: #f00;
  }
</style>
```



```
<!-- ButtonContainer.svelte -->
<script>
  import MyButton from './MyButton.svelte'
</script>

<MyButton />
<button>Another button</button>

<style>
  button {
    color: #00f;
  }
</style>
```

Templating

Conditionals

```svelte
<script lang="ts">
  let theme = 'light'
</script>

{#if theme === 'light'}
  light theme
{:else}
  dark theme
{/if}
```

Loops

```
<script>
  let fruits = ['apple', 'pear','banana']
</script>

<ul>
  {#each fruits as fruit}
    <li>{fruit}</li>
  {/each}
</ul>
```

Reactivity

```
<script>
  let count = 0

  const handleClick = () => count++
</script>

<button on:click={handleClick}>
  clickcount: {count}
</button>
```

How does this work though?

Reactivity

```
<script>
  let count = 0
  const handleClick = () => count+

  $: doubleCount = count * 2
</script>

<button on:click={handleClick}>
  clickcount: {count}
</button>

<h1>doubleCount:> {doubleCount}</h
```

```
// tangent: the $: thing

loop1:
for (i = 0; i < 3; i++) {
  loop2:
  for (j = 0; j < 3; j++) {
    if (...) {
      continue loop1;
    }
    ...
  }
}
```

How does this work though?

(and what's this weird $: thing...)

Reactivity Statements

```
$: console.log('User made' + count + 'clicks');
```

```
$: {
  console.log('Current score: ' + count);
  alert('Score with premium: ' + count);
}
```

```
$: if (count > 10) {
  console.log('ten is the maximum sorry');
  count = 10;
}
```

Events

DOM Events

```
<script>
  let m = { x: 0, y: 0 };

  function handleMousemove(event) {
    m.x = event.clientX;
    m.y = event.clientY;
  }
</script>

<div on:mousemove={handleMousemove}>
  The mouse position is {m.x} x {m.y}
</div>
```

All regular DOM events can be used, prefixed with the `:on` keyword.

Component Lifecycle Hooks

Execute certain actions during the lifecycle of a component

Possible use cases:

- load data from a backend
- start a timer
- cleanup resources

onMount / onDestroy

```
<script>
  import { onMount, onDestroy } from 'svelte'

  let interval = null
  let elapsedTime = 0
  onMount(() => {
    interval = setInterval(() => elapsedTime++, 1000)
  })

  onDestroy(() => clearInterval(interval))
</script>

elapsedTime: {elapsedTime} second(s)
```

onMount / onDestroy

Instead of using onDestroy just return a method from onMount

```
<script>
  import { onMount } from 'svelte'

  let elapsedTime = 0
  onMount(() => {
    const interval = setInterval(() => elapsedTime++, 1000)

    return () => clearInterval(interval)
  })
</script>

elapsedTime: {elapsedTime} second(s)
```

There are a other hooks like `beforeUpdate`, `afterUpdate`, but they are not as relevant.

(Global) State Handling

- built-in
- holds (applicationwide) data
- means of communication (messaging) between application parts

Writable Store

```
import { writable } from 'svelte/store'

const fruitStore = writable<string[]>([])

// subscribe to updates
fruitStore.subscribe(fruits => {
  console.log(fruits)
})

// set the value of the store
fruitStore.set(['apple', 'banana', 'pear'])

// update the value in the context of existing values
fruitStore.update(currentFruits => {
  return [...currentFruits, 'grapes']
})
```

Render Store Data

```svelte
<script lang="ts">
  import { onDestroy } from 'svelte'
  import fruitStore from './store'

  let myFruits = []
  const unsubscribe = fruitStore.subscribe(fruitState => {
    myFruits = fruitState
  })

  onDestroy(unsubscribe)
</script>

{#each myFruits as fruit}
  {fruit}<br>
{/each}
```

Store Auto Subscriptions

```
<script lang="ts">
  import fruitStore from './store'
</script>

{#each $fruitStore as fruit}
  {fruit}<br>
{/each}
```

How does this work though?

Transitions / Animations

Transitions

```
<script>
  let visible = false
</script>

<button on:click={() => visible = !visible}>
  toggle
</button>

{#if visible}
  <h1>ohey!</h1>
{/if}
```

Problem: A declarative template does not account for in between states.

How can we fade between states?

Transitions

```
<script>
  import { fade } from 'svelte/transition'

  let visible = false
</script>

<button on:click={() => visible = !visible}>
  toggle
</button>

{#if visible}
  <h1 transition:fade>ohey!</h1>
{/if}
```

Transitions

- 7 built-in transitions: `fade`, `blur`, `fly`, `slide`, `scale`, `draw`, `crossfade`
- In and out transitions can be declared separately
- Transitions can be parametrized
- Custom transitions

```
<p in:fly="{{ y: 200, duration: 2000 }}" out:fade>
  Flies in, fades out
</p>
```

Animations

Animations

```
<script>
  import { tweened } from 'svelte/motion'

  const position = tweened(0, { duration: 800 })

  const moveLeft = () => position.set(0)
  const moveRight = () => position.set(80)
</script>

<div>
  <div style="left: {$position}%"></div>
</div>

<button on:click={moveLeft}>Left</button>
<button on:click={moveRight}>Right</button>
```

Animations

`tweened` also accepts an easing parameter

```
<script>
  import { tweened } from 'svelte/motion'
  import { cubicOut } from 'svelte/easing';

  const position = tweened(0, {
    duration: 800,
    easing: cubicOut
  })
</script>
```

There are a couple of built-in easing methods:
`back`, `bounce`, `circ`, `cubic`, `elastic`, `expo`, `quad`, `quart`, `quint`, `sine`

Each as an *in*, an *out*, and an *inOut* variant.

Resources

- Tutorial: https://svelte.dev/tutorial
- Documentation: https://svelte.dev/docs
- Sveltekit: https://kit.svelte.dev