Svelte:
Today / Tomorrow

inovex

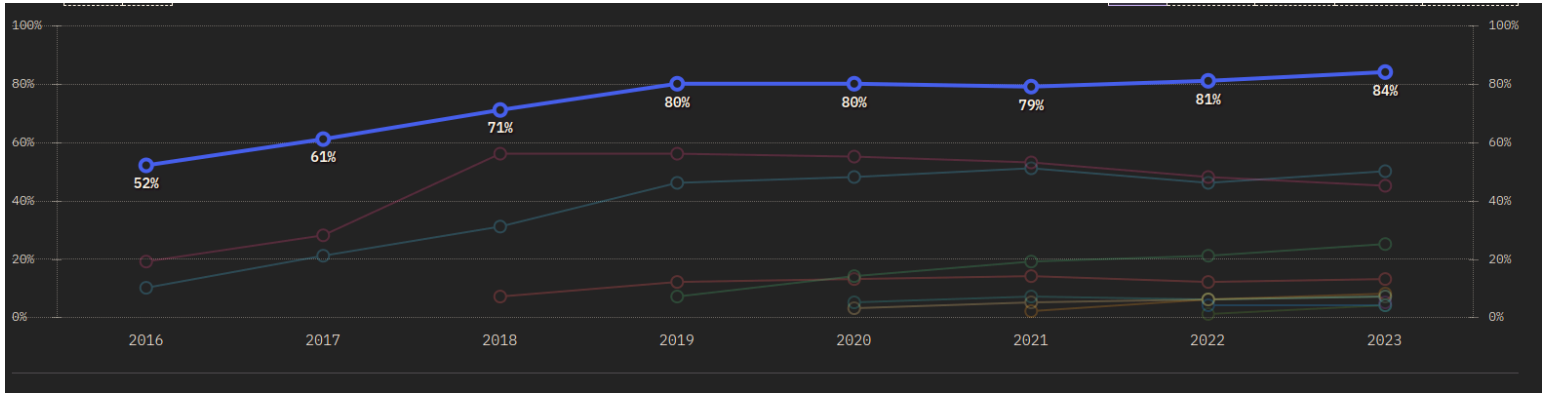
Manuel Ernst

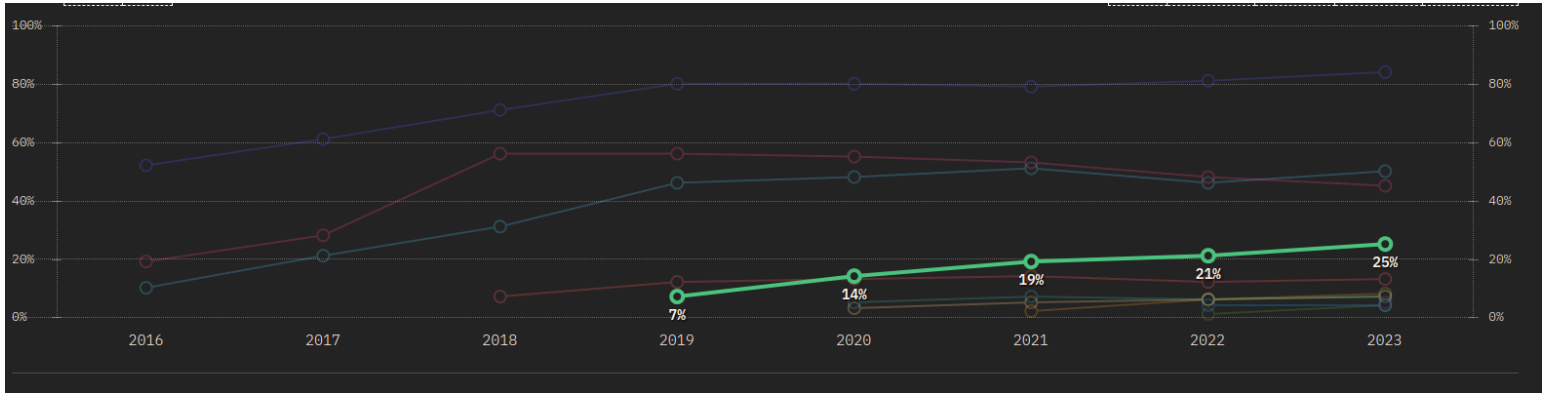
- Software Engineer at inovex Erlangen
- Web-Development for over 20 years

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

@seriousManual

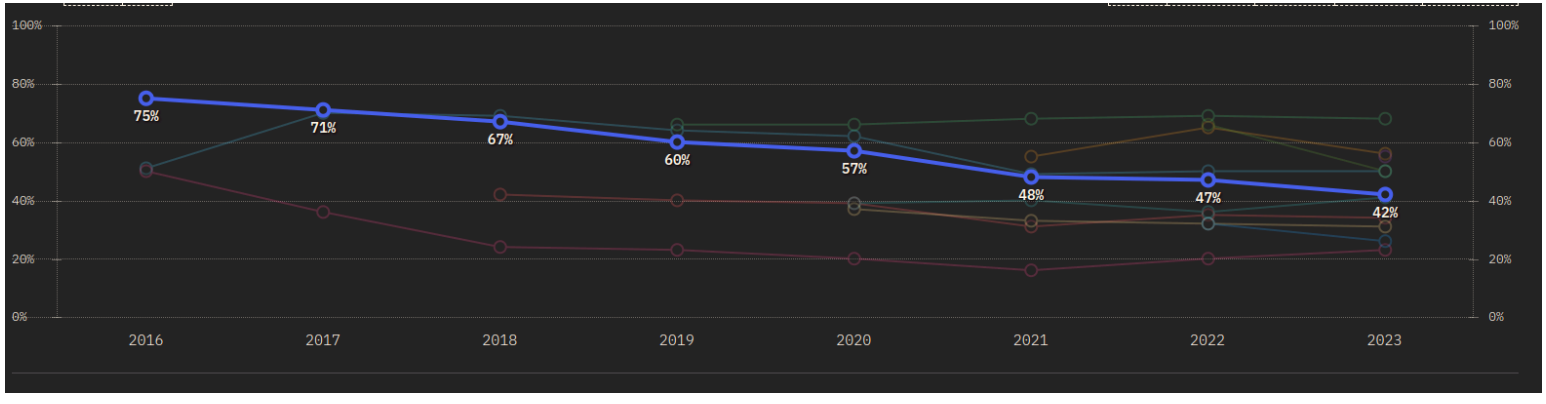
Usage

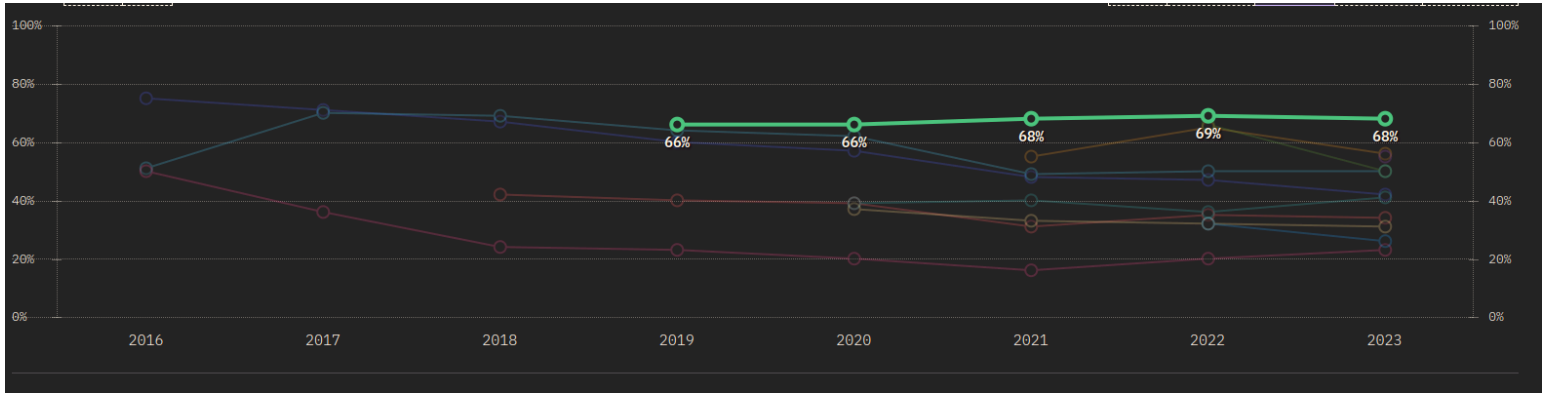




Source: *State Of JS 2023*

Interest

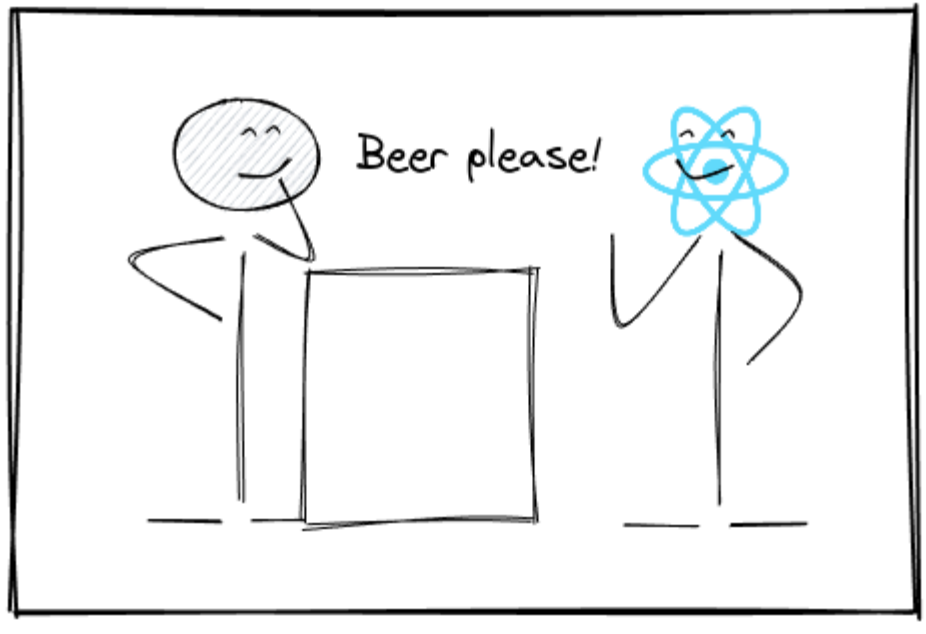




Source: *State Of JS 2023*

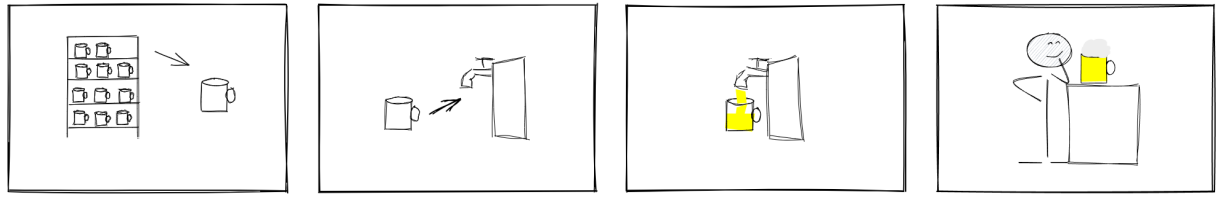
Popular Svelte Users

Imperative vs. Declarative



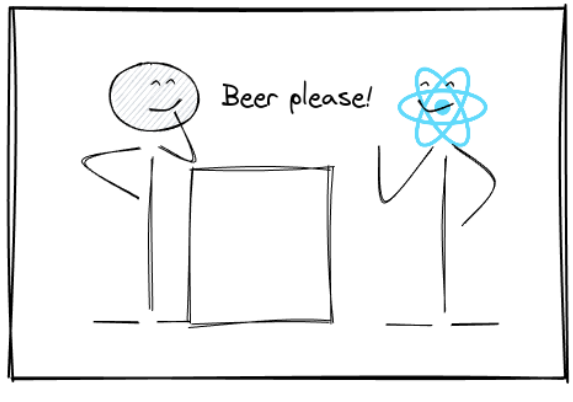
Imperative UI Definition

```
1  let counter = 0
2
3  const button = document.createElement('button')
4  button.innerText = `clickcount: ${counter}`
5
6  button.addEventListener('click', () => {
7    counter++
8    button.innerText = `clickcount: ${counter}`
9  })
10
11 body.appendChild(button)
```

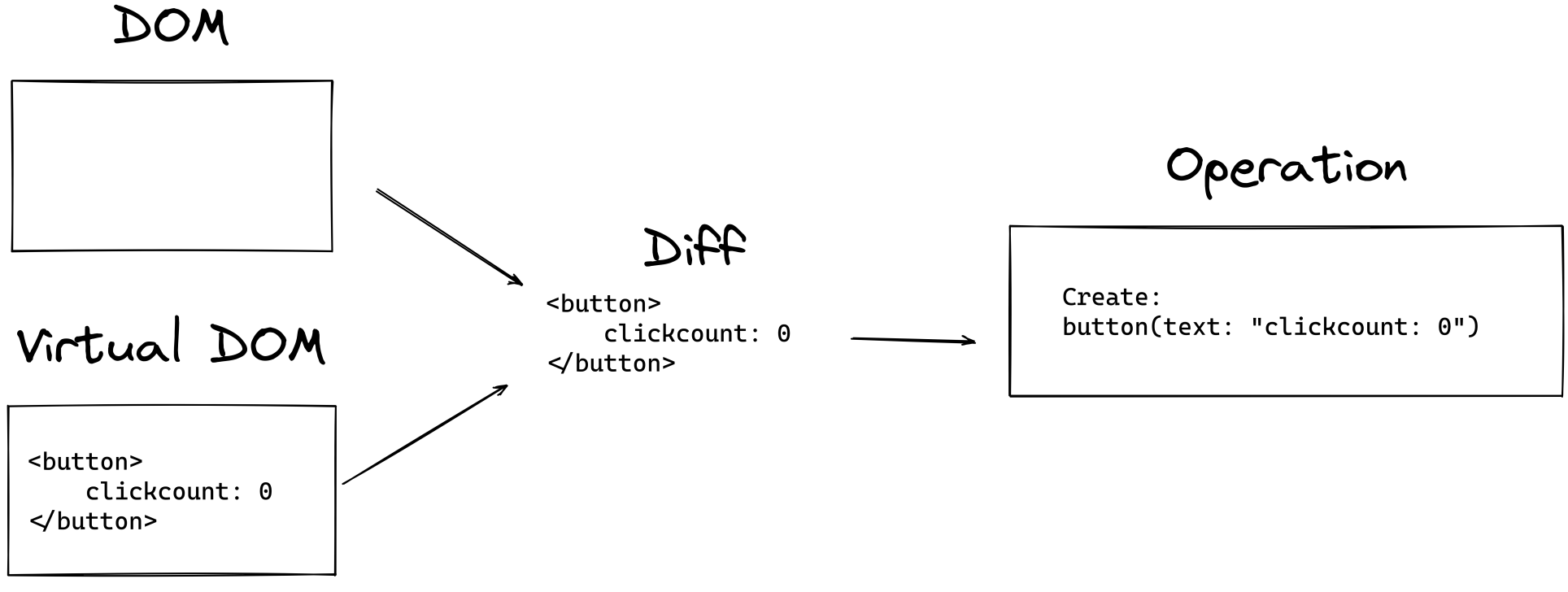


Declarative UI Definition

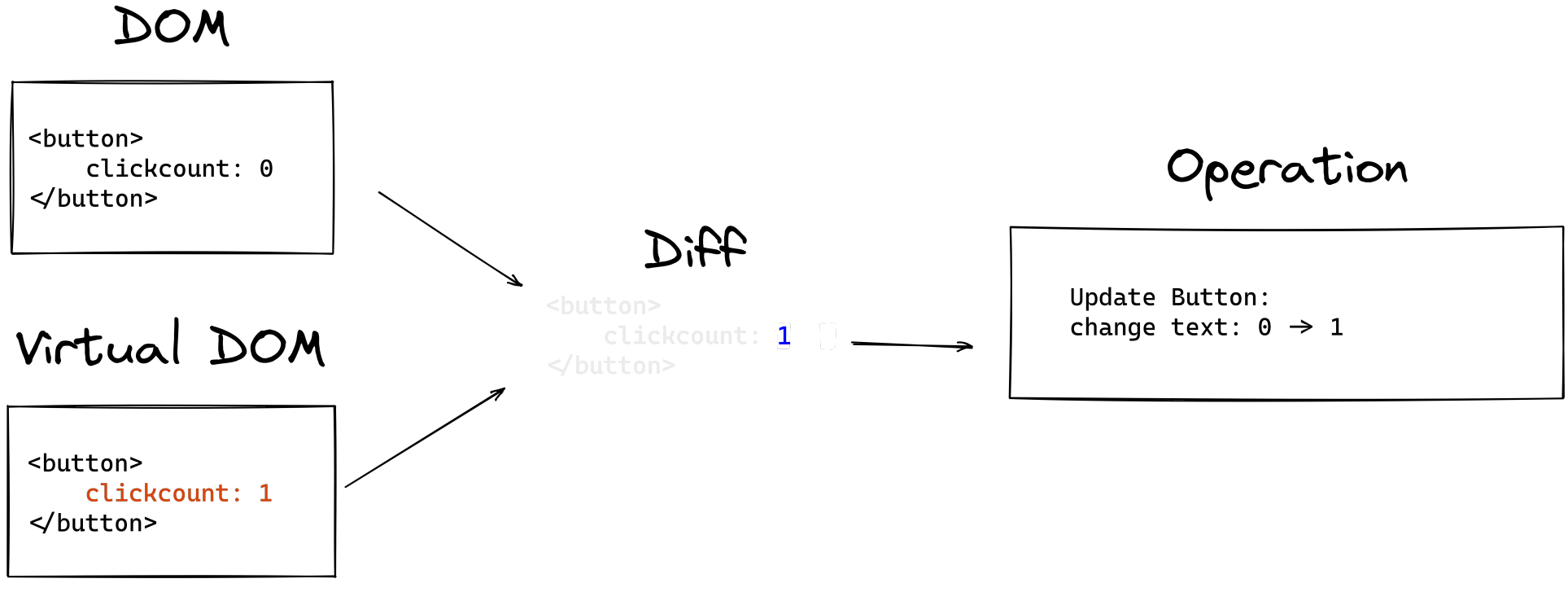
```
1 function Counter() {
2   const [count, setCount] = useState(0)
3
4   return (
5     <button onClick={() => setCount(count + 1)}>
6       clickcount: {count}
7     </button>
8   )
9 }
```



Runtime DOM Diffing



Runtime DOM Diffing



Svelte - Components

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

- Markup
- Styling
- Logic

Svelte - Components

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

Output

Hello world!

Templating

```
1 <script lang="ts">
2   const name = 'world'
3 </script>
4
5 <h2>Hello {name}!</h2>
```

Output

Hello world!

Dynamic Attributes

```
1 <script lang="ts">
2   const src = '/tutorial/image.gif'
3 </script>
4
5 <img src={src}>
```

Output

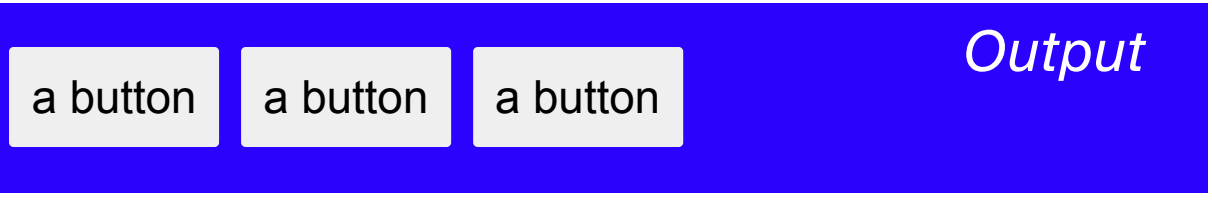
```
1 <!-- short version -->
2 <img {src}>
```

Referencing Components

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

```
1 <!-- ButtonContainer.svelte -->
2 <script>
3   import MyButton from './MyButton.svelte'
4 </script>
5
6 <MyButton />
7 <MyButton />
8 <MyButton />
```

Output



Props for Components

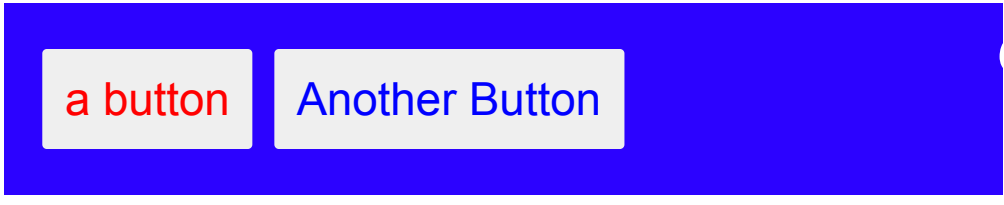
```svelte
<!-- MyButton.svelte -->
<script>
  export let caption = 'a button'
</script>

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

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

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

Output

Styling

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

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




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

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

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

Templating

Conditionals

```
<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>
```

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.

Reactivity

```
1 <script>
2   let count = 0
3
4   const increment = () => count++
5 </script>
6
7 <button on:click={increment}>
8   clickcount: {count}
9 </button>
```

How does Svelte mirror these changes in the DOM?

```
import React, { useState } from 'react';

function Example() {
    const [count, setCount] = useState(0);
}
```

```
let count = 0;
```

Reactivity

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

  $: doubleCount = count * 2
</script>

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

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

```
// 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...)

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.