INTRO, ACTIONS AND STORES

SVELTE

INTRODUCTION

WHAT IS IT?

- A single file component framework
- Built-in Features
- Opinionated

WHY?

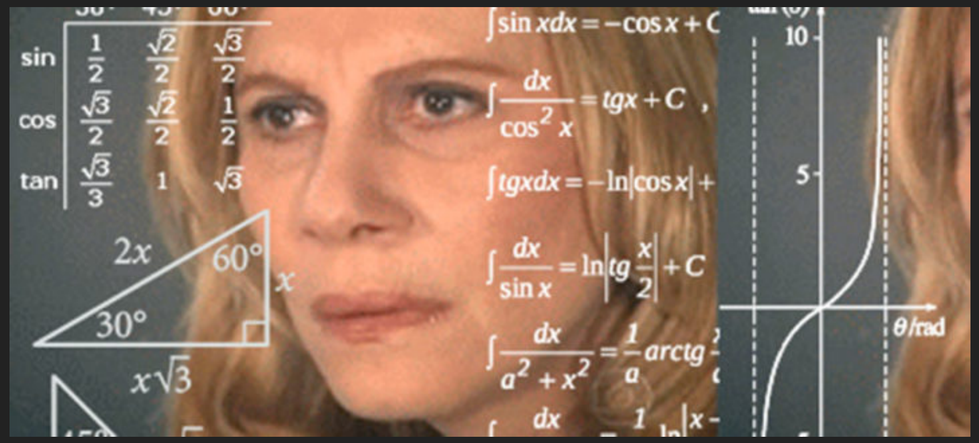
- Simple
- Fast
- Great DX

```
<script>
  export let image;
</script>

<style lang="scss">
  .avatar {
    display: grid;
    border: solid var(--white) 2px;
    border-radius: 60px;
    overflow: hidden;
    img {
      width: 60px;
      height: 60px;
    }
  }
</style>

<div class="avatar">
  <img src={image} alt="profile image" />
</div>
```

HOW DOES IT WORK?

- Compiler
- No virtual DOM
- Updates Values

- Little Overhead
- No re-rendering of components
- Features can be tree-shaken



PROPS AND REACTIVITY

```svelte
<script>
  export let firstName = "Adam"
  export let lastName = "Svensson"

  // Reactive Declaration
  $: fullName = `${firstName} ${lastName}`

</script>

<p>{fullName}</p>

```

```svelte
<NameComponent firstName="Kevin" lastName="Åberg"/>
```

UPDATING ARRAYS, AND EVENTS

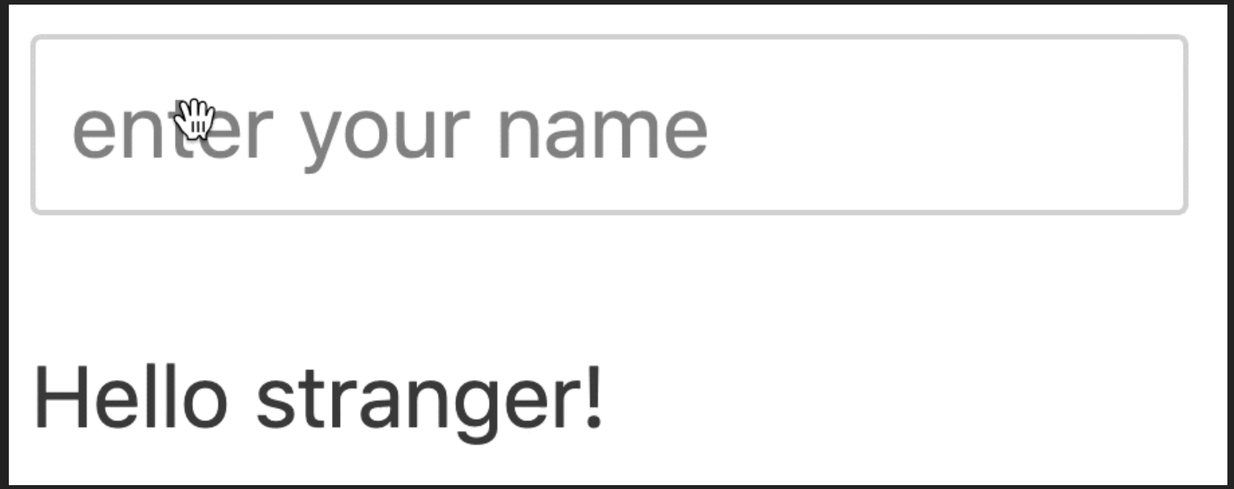
```svelte
<script>
  let numbers = [1, 2, 3, 4];

  function addNumber() {
    // Will not update the DOM, but update the array
    // numbers.push(numbers.length + 1);

    numbers = [...numbers, numbers.length + 1];
  }

  $: sum = numbers.reduce((t, n) => t + n, 0);
</script>

<p>{numbers.join(' + ')} = {sum}</p>

<button on:click={addNumber}>
  Add a number
</button>

```

MORE ON EVENTS

```
<script>
  function handleClick() {
    alert('only run once')
  }
</script>

<button on:click|once={handleClick}>
  Click me
</button>
```

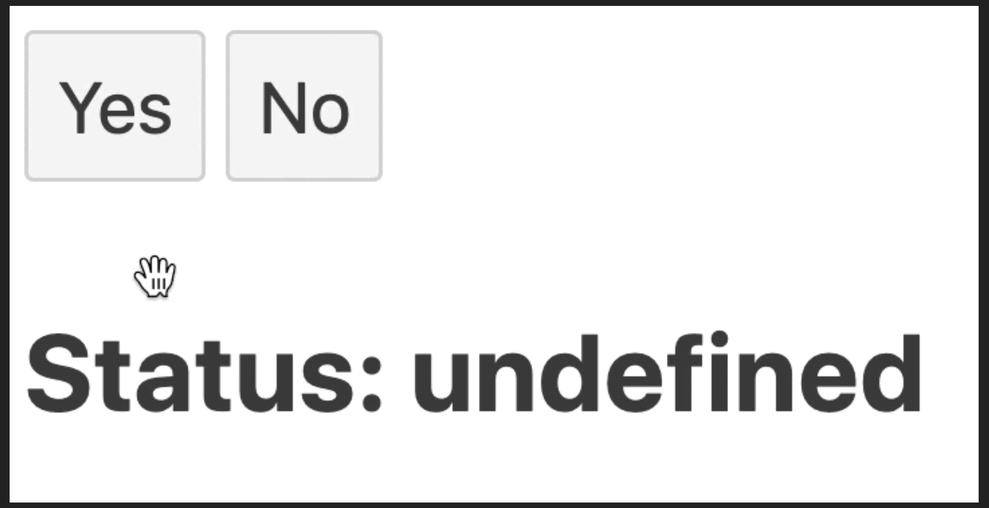
INPUT COMPONENT & BIND

```
<script>
  let name = '';
</script>

<input bind:value={name} placeholder="enter your name">
<p>Hello {name || 'stranger'}!</p>
```



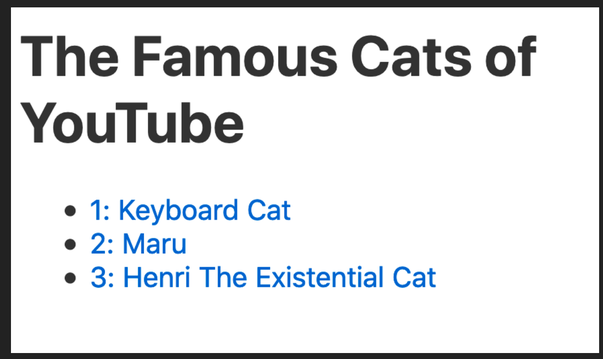
LOGIC 1 - IF

```svelte
<script>
  let user = { status: undefined };
</script>

<button on:click={() => (user.status = true)}>Yes</button>
<button on:click={() => (user.status = false)}>No</button>

{#if user.status === undefined}
  <h2>Status: {user.status}</h2>
{:else if user.status}
  <h2>Welcome!</h2>
{:else}
  <h2>Goodbye!</h2>
{/if}
```



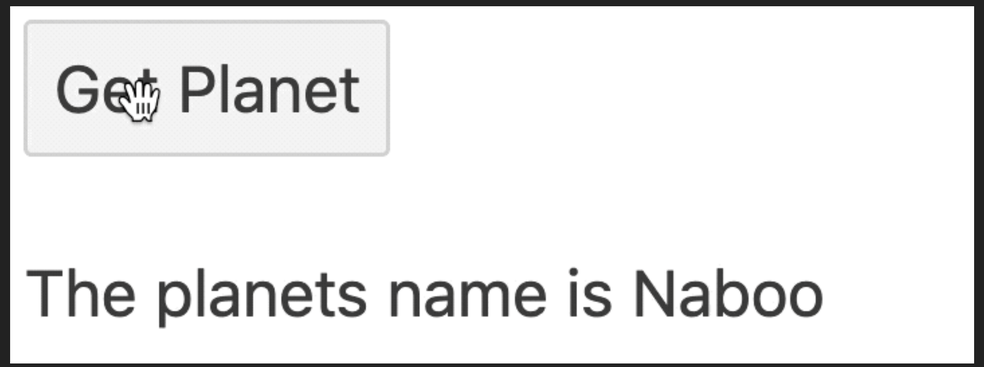
LOGIC 2 - EACH

```svelte
<script>
  let cats = [
    { id: 'J---aiyznGQ', name: 'Keyboard Cat' },
    { id: 'z_AbfPXTKms', name: 'Maru' },
    { id: 'OUtn3pvWmpg', name: 'Henri The Existential Cat' }
  ];
</script>

<h1>The Famous Cats of YouTube</h1>

<ul>
  {#each cats as { id, name }, i}
    <li><a target="_blank" href="https://www.youtube.com/watch?v={id}">
      {i + 1}: {name}
    </a></li>
  {/each}
</ul>
```



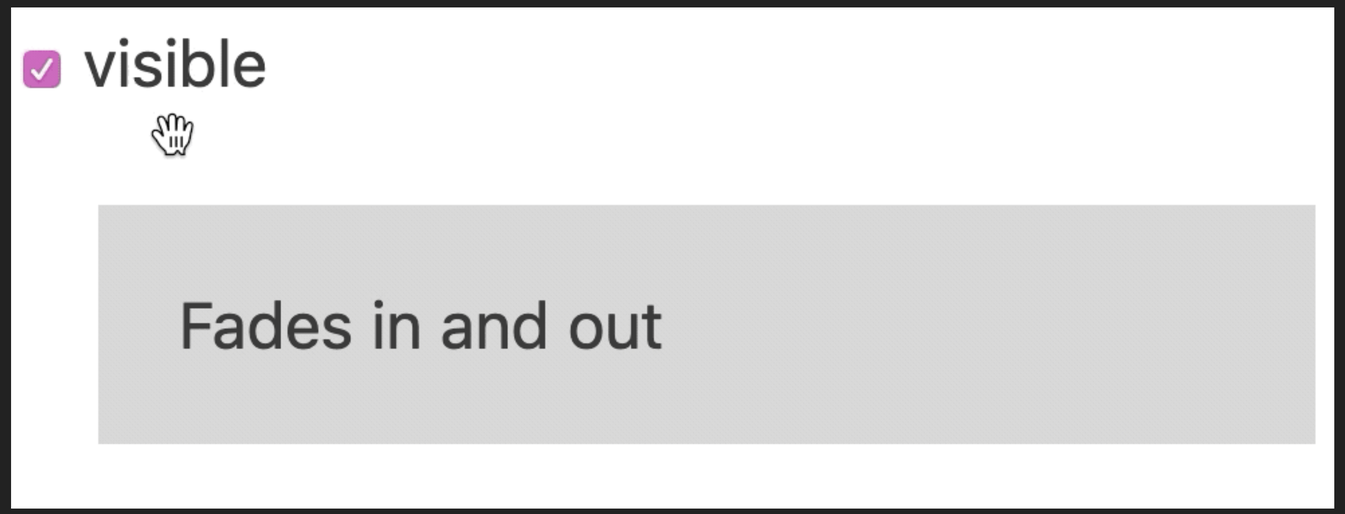
LOGIC 3 – AWAIT

```svelte
<script>
  let promise = getData();

  async function getData() {
    const randomNumber = Math.floor((Math.random() * 10) + 1)
    const res = await fetch(`https://swapi.co/api/planets/${randomNumber}/`);
    const text = await res.json();

    return text
  }

  function handleClick() {
    promise = getData();
  }
</script>

<button on:click={handleClick}>
  Get Planet
</button>

{#await promise}
  <p>...waiting</p>
{:then planet}
  <p>The planets name is {planet.name}</p>
{:catch error}
  <p style="color: red">{error.message}</p>
{/await}
```



TRANSITIONS

```svelte
<script>
  import { fade } from 'svelte/transition';
  let visible = true;
</script>

<label>
  <input type="checkbox" bind:checked={visible}>
  visible
</label>

{#if visible}
  <p transition:fade>
    Fades in and out
  </p>
{/if}


<style>
  p {
    background: lightgrey;
    padding: 20px;
    margin: 20px;
  }
</style>
```



MORE ON TRANSITIONS

ACTIONS

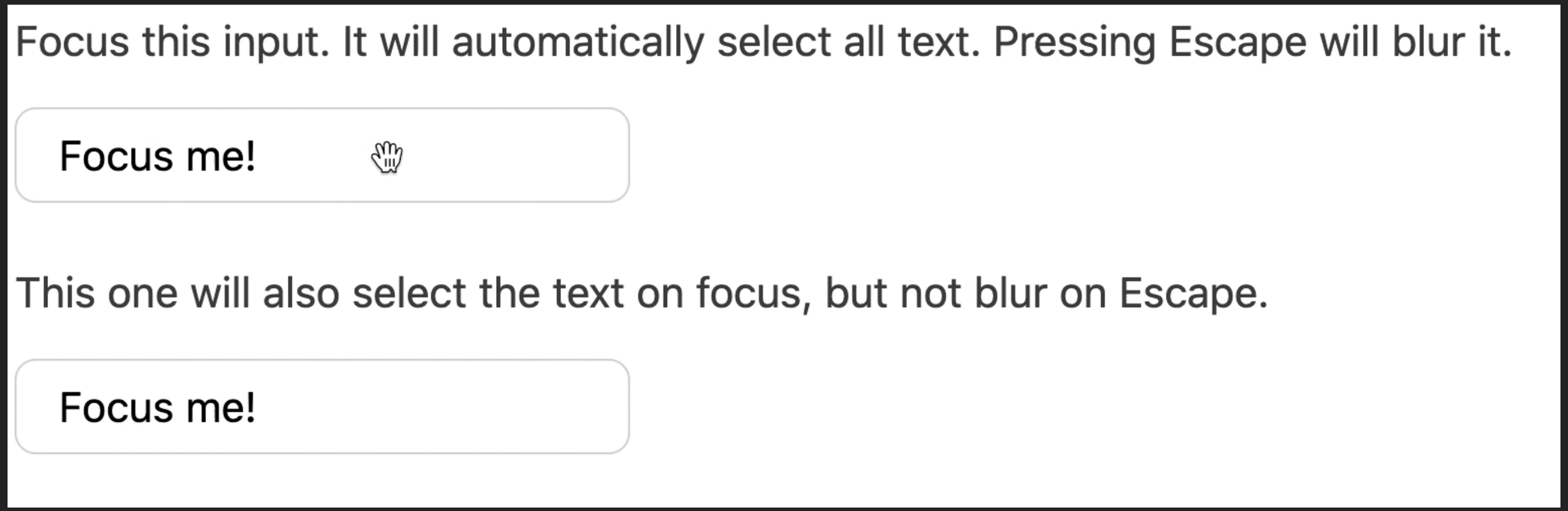
WHAT ARE THEY?

- Simple Functions
- Extracted functionality

WHAT DOES IT LOOK LIKE?

```
<script>
  function foo(node) {
    // the node has been mounted in the DOM

    return {
      destroy() {
        // the node has been removed from the DOM
      }
    };
  }
</script>

<div use:foo></div>
```

EXAMPLE – FOCUS TEXT INPUT (1/2)



EXAMPLE – FOCUS TEXT INPUT (2/2)

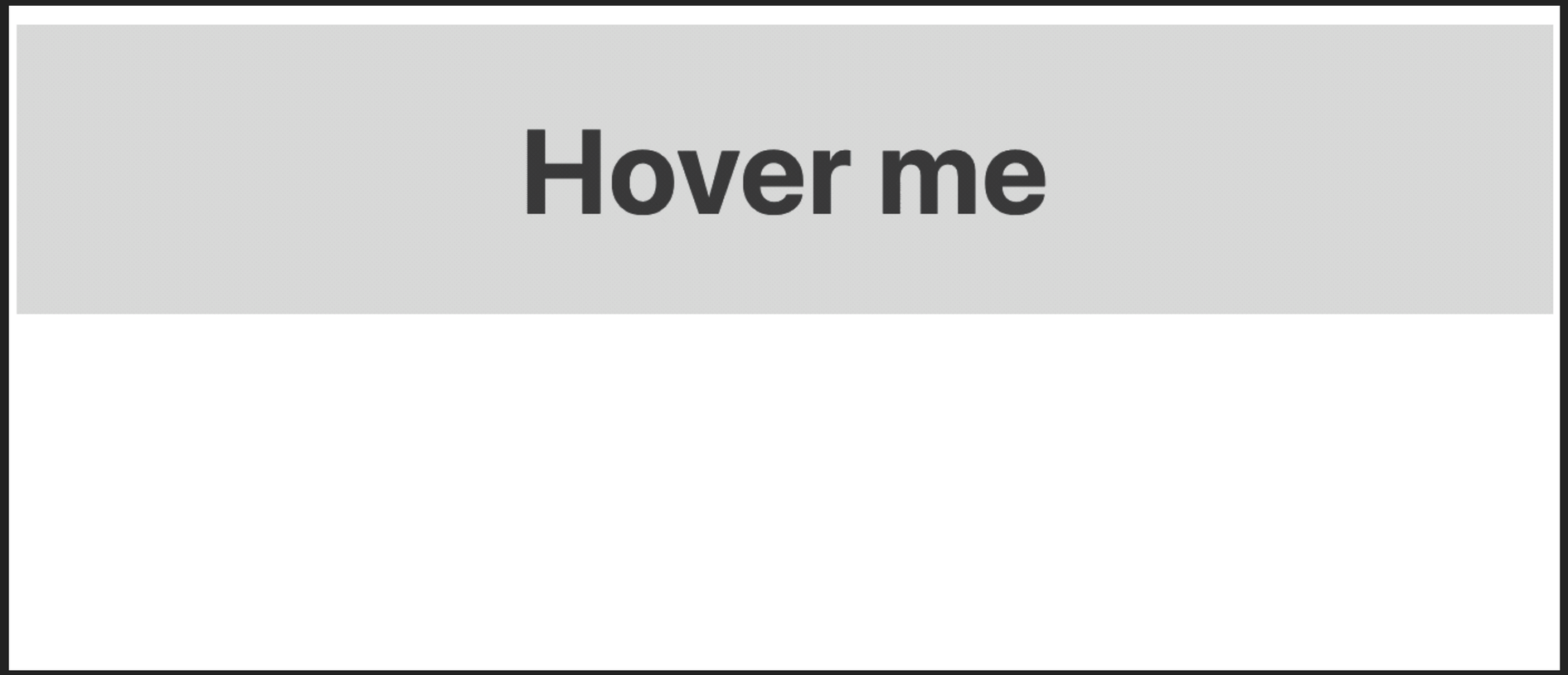
App.svelte

```svelte
<script>
  import {selectTextOnFocus, blurOnEscape} from './inputDirectives.js';

  let value = 'Focus me!';
</script>

<p>Focus this input. It will automatically select all text. Pressing Escape will blur it.</p>
<input bind:value={value} use:selectTextOnFocus use:blurOnEscape/>

<p>This one will also select the text on focus, but not blur on Escape.</p>
<input bind:value={value} use:selectTextOnFocus/>

<style>
  input {
    border-radius: .5em;
    padding: .5em 1em;
    outline: none;
  }

  input:focus {
    border-color: steelblue;
  }
</style>

```

inputDirectives.js

```js
/** Selects the text inside a text node when the node is focused */
export function selectTextOnFocus(node) {

  const handleFocus = event => {
    node && typeof node.select === 'function' && node.select()
  }

  node.addEventListener('focus', handleFocus)

  return {
    destroy() {
      node.removeEventListener('focus', handleFocus)
    }
  }
}

/** Blurs the node when Escape is pressed */
export function blurOnEscape(node) {

  const handleKey = event => {
    if (event.key === 'Escape' && node && typeof node.blur === 'function') node.blur()
  }

  node.addEventListener('keydown', handleKey)

  return {
    destroy() {
      node.removeEventListener('keydown', handleKey)
    }
  }
}

```

EXAMPLE – TOOLTIP (1/2)



EXAMPLE – TOOLTIP (2/2)

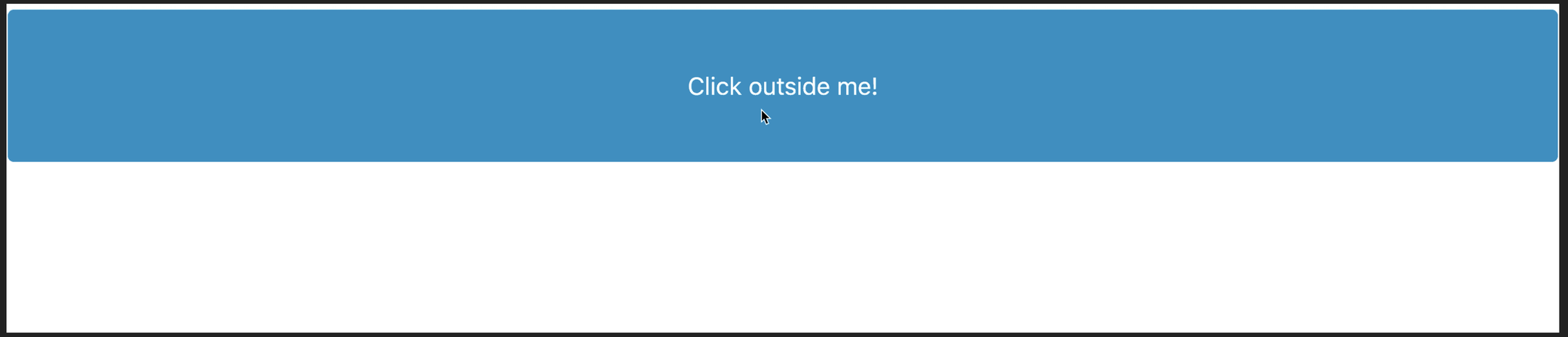
App.svelte

```
<script>
  import Tooltip from './Tooltip.svelte'
  import { tooltip } from './tooltip.js'
</script>

<style>
  h1 {
    padding: 20px;
    text-align: center;
    position:relative;
    background:lightgrey;
  }
</style>

<h1 use:tooltip={{content: Tooltip}}>
  Hover me
</h1>


```

tooltip.js

```
export function tooltip(node, options) {

  let component, hover;

  node.addEventListener('mouseover', attachTooltip);
  node.addEventListener('mouseout', removeTooltip);

  function attachTooltip() {
    component = new options.content({target: node})
  }

  function removeTooltip() {
    component.$destroy()
  }

  return {
    destroy() {
      node.removeEventListener('mouseover', attachTooltip);
      node.removeEventListener('mouseout', removeTooltip);
    }
  };
}

```

EXAMPLE – CLICK OUTSIDE (1/2)



EXAMPLE – CLICK OUTSIDE (2/2)

App.svelte

```svelte
<script>
  import {clickOutside} from './clickOutside.js';

  function handleClickOutside(event) {
    alert('Click outside!');
  }

</script>

<div use:clickOutside on:click_outside={handleClickOutside}>
  Click outside me!
</div>

<style>
  div {
    height: 100px;
    color: white;
    background-color: steelblue;
    border-radius: 4px;
    display: flex;
    align-items:center;
    justify-content: center;
  }
</style>
```

clickOutside.js

```js
/** Dispatch event on click outside of node */
export function clickOutside(node) {

  const handleClick = event => {
    if (node && !node.contains(event.target) && !event.defaultPrevented) {
      node.dispatchEvent(
        new CustomEvent('click_outside', node)
      )
    }
  }

  document.addEventListener('click', handleClick, true);

  return {
    destroy() {
      document.removeEventListener('click', handleClick, true);
    }
  }
}
```

STORES

WHAT ARE THEY?

- State management
- Global
- Easy to use yet powerful

WHAT DOES IT LOOK LIKE?

```svelte
<script>
  import { writable } from 'svelte/store';

  const count = writable(0);
  console.log($count); // logs 0

  count.set(1);
  console.log($count); // logs 1

  $count = 2;
  console.log($count); // logs 2
</script>
```

EXAMPLE – A SIMPLE ONE (1/2)

App.svelte

```
<script>
  import Red from './Red.svelte'
  import Blue from './Blue.svelte'
  import { number } from './number.js';
</script>

<h1>Toggle number:</h1>
<button on:click={() => $number = $number * 2}>
  Double
</button>

<Red />
<Blue />
```

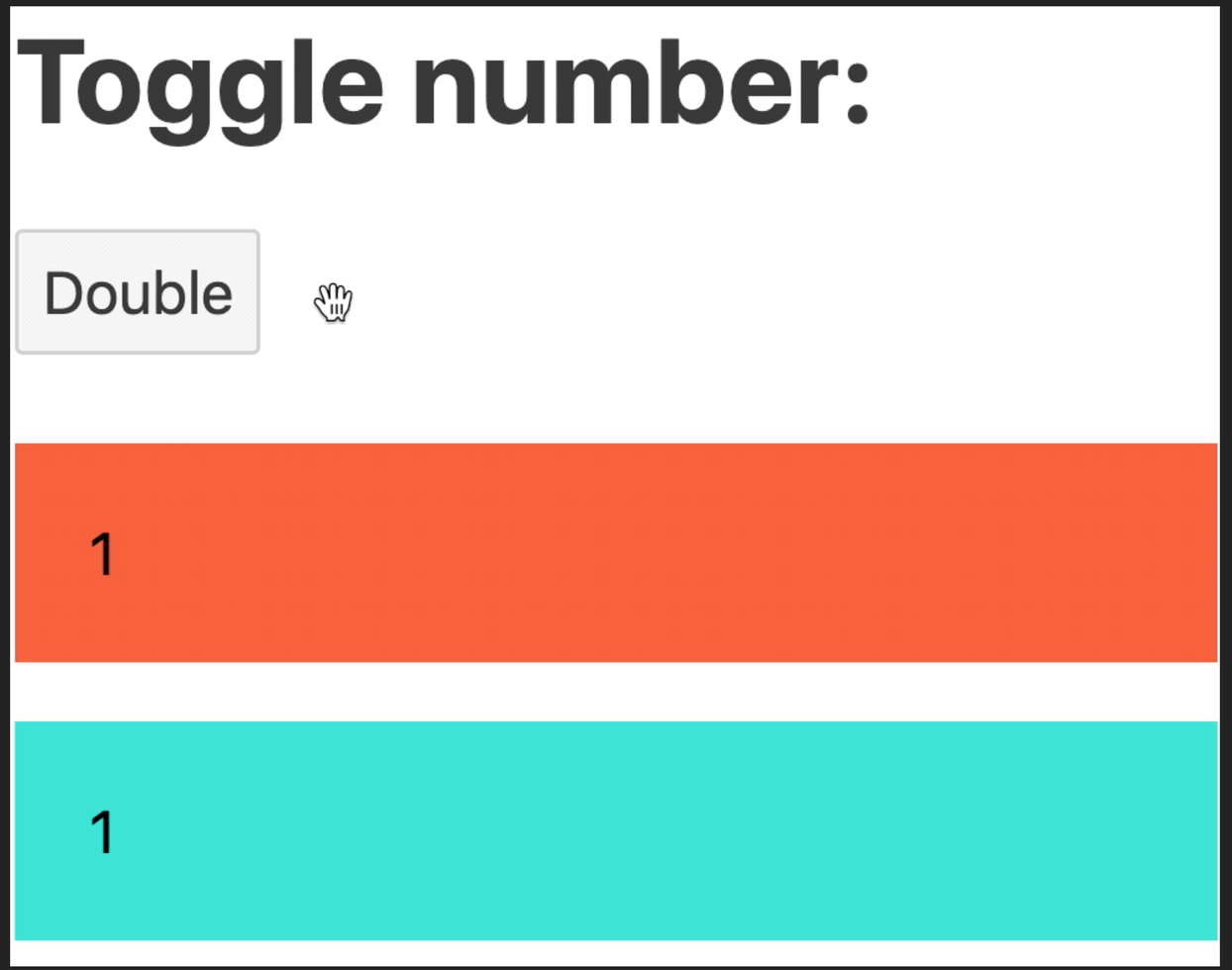
number.js

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

export const number = writable(1);
```

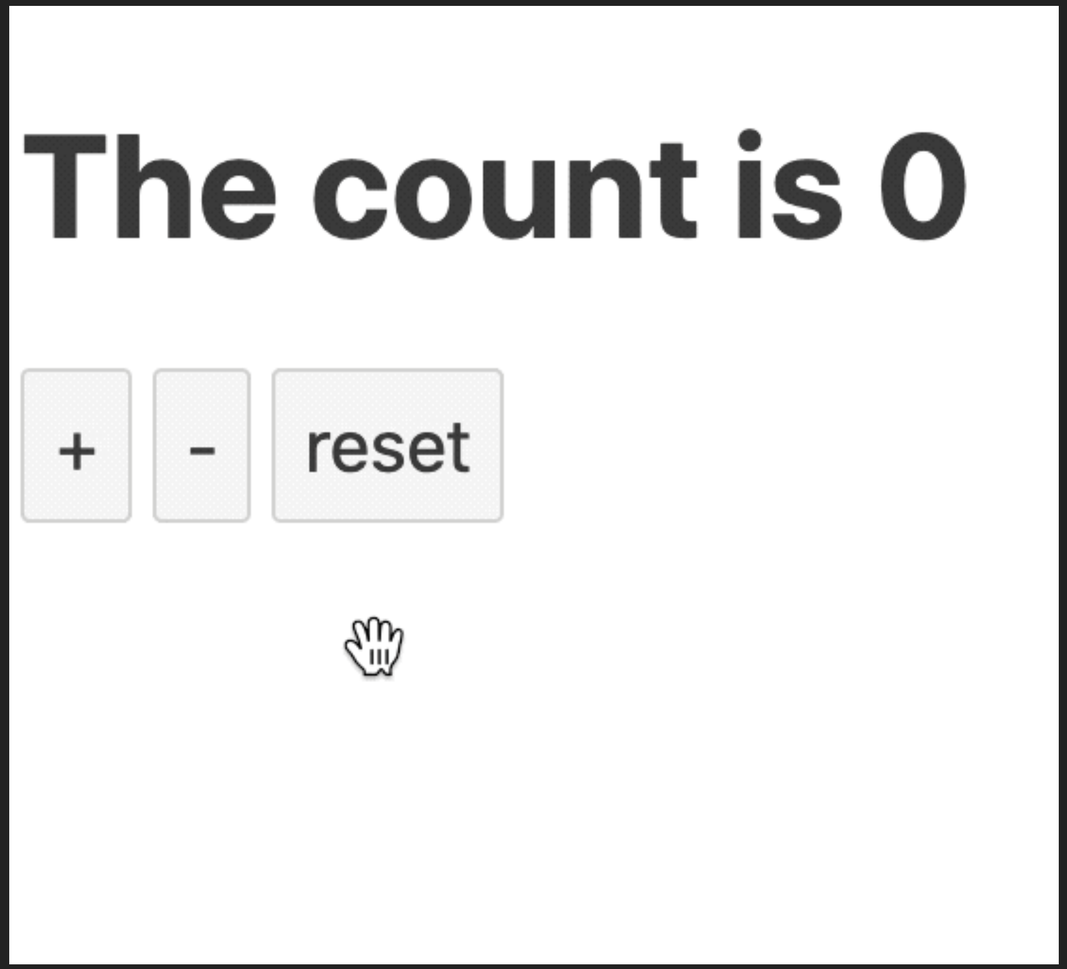
Red.svelte or Blue.svelte

```
<script>
  import { number } from './number.js';
</script>

<p>
  {$number}
</p>
```

EXAMPLE – A SIMPLE ONE (2/2)



EXAMPLE – CUSTOM STORE (1/2)



EXAMPLE – CUSTOM STORE (2/2)

App.svelte

```svelte
<script>
  import { count } from './stores.js';
</script>

<h1>The count is {$count}</h1>

<button on:click={count.increment}>+</button>
<button on:click={count.decrement}>-</button>
<button on:click={count.reset}>reset</button>

```

stores.js

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

function createCount() {
  const { subscribe, set, update } = writable(0);

  return {
    subscribe,
    increment: () => update(n => n + 1),
    decrement: () => update(n => n - 1),
    reset: () => set(0)
  };
}

export const count = createCount();

```

EXAMPLE – RXJS INTEROPERABILITY

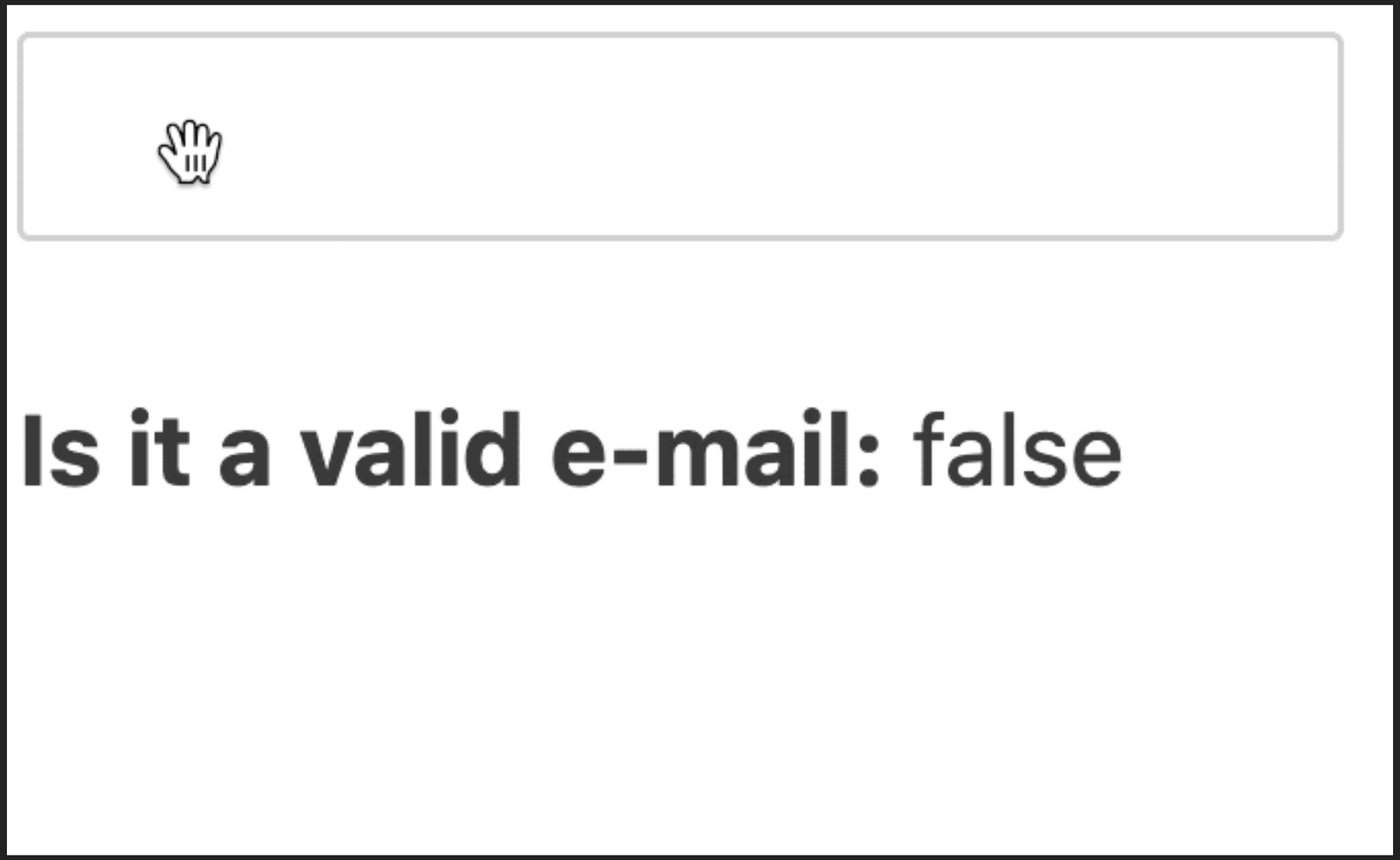
App.svelte

```svelte
<script>
  import { timer$ } from './store.js'
</script>

<h1>RxJS Example</h1>

{$timer$}
```

store.js

```js
import { interval } from 'rxjs'
import { map } from 'rxjs/operators'

export const timer$ = interval(1000).pipe(map(value => value * 2))
```

EXAMPLE – ACTIONS AND STORES (1/2)



EXAMPLE – ACTIONS AND STORES (2/2)

App.svelte

```svelte
<script>
  import { email } from './regex.js';
  import { validator } from './validator.js';

  let [ isValid, validate ] = validator(email);
</script>

<input use:validate>

<p><strong>Is it a valid e-mail:</strong> {$isValid}</p>


```

validator.js

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

export function validator(regex, initial = false) {
  const { subscribe, set } = writable(initial);

  const action = (node) => {
    function _v(e) {
      set(regex.test(e.target.value))
    }
    node.addEventListener('change', _v)
    return { destroy: () => node.removeEventListener('change', _v) }
  }

  return [ { subscribe }, action ]
}

```

SUMMARY

- Lots of built-in features at no cost
- Actions: Advanced functionality at your fingertips
- Stores: State management