



# A world outside Polymer

Horacio Gonzalez  
@LostInBrittany

# Horacio Gonzalez



@LostInBrittany

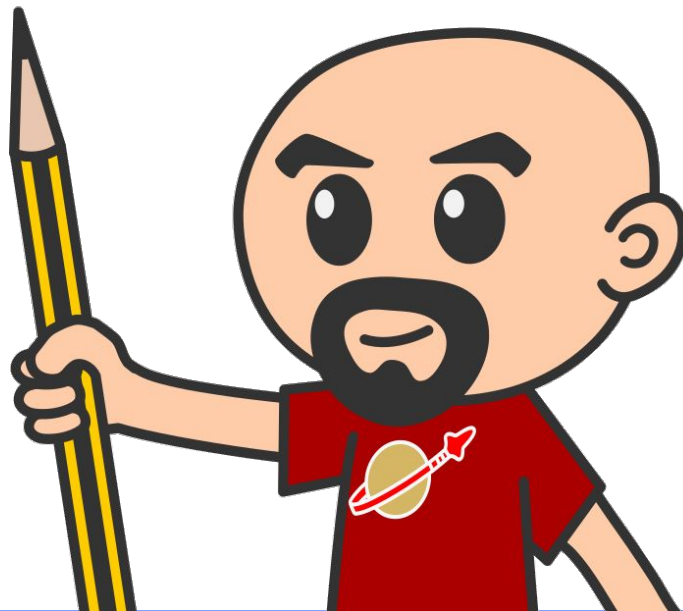
Spaniard lost in Brittany,  
developer, dreamer and  
all-around geek



<http://cityzendata.com>



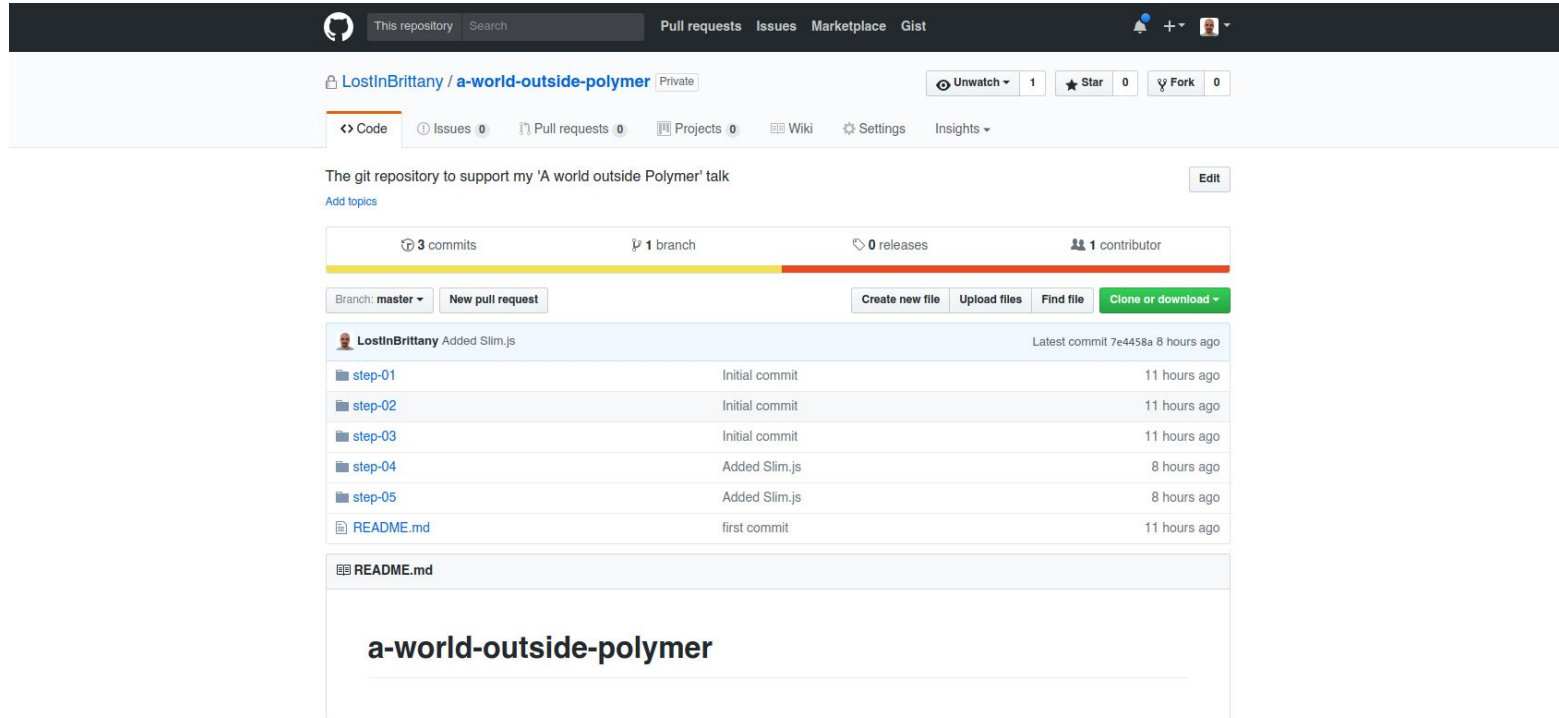
**WARP 10**



#RennesJS #Polymer #MixingWebComponents

@LostInBrittany

# We want the code!



The screenshot shows a GitHub repository page for 'a-world-outside-polymer' by user 'LostInBrittany'. The repository is private and has 1 star, 0 forks, and 0 issues. It contains 3 commits, 1 branch, and 0 releases. The commit history shows a series of steps: 'step-01' through 'step-05' and 'README.md'. The 'step-04' and 'step-05' commits are labeled 'Added Slim.js'. The 'README.md' file content is visible, showing the repository name 'a-world-outside-polymer'.

Commit	Message	Time
Latest commit 7e4458a	Added Slim.js	8 hours ago
step-01	Initial commit	11 hours ago
step-02	Initial commit	11 hours ago
step-03	Initial commit	11 hours ago
step-04	Added Slim.js	8 hours ago
step-05	Added Slim.js	8 hours ago
README.md	first commit	11 hours ago

<https://github.com/LostInBrittany/a-world-outside-polymer>

# Web Components

---



# A very basic web component

```
class MyElement extends HTMLElement {  
  
  // This gets called when the HTML parser sees your tag  
  constructor() {  
    super(); // always call super() first in the ctor.  
    this.msg = 'Hello, RennesJS!';  
  }  
  
  // Called when your element is inserted in the DOM or  
  // immediately after the constructor if it's already in the DOM  
  connectedCallback() {  
    this.innerHTML = `

${this.msg}</p>`;  
  }  
}  
  
customElements.define('my-element', MyElement);


```

# Custom Elements:

- Let you define your own HTML tag with bundled JS behavior
- Trigger lifecycle callbacks
- Automatically “upgrade” your tag when inserted in the document

# Custom Elements don't:

- Scope CSS styles
  - Shadow DOM
- Scope JavaScript
  - ES2015
- “Reproject” children into `<slot>` elements
  - Shadow DOM

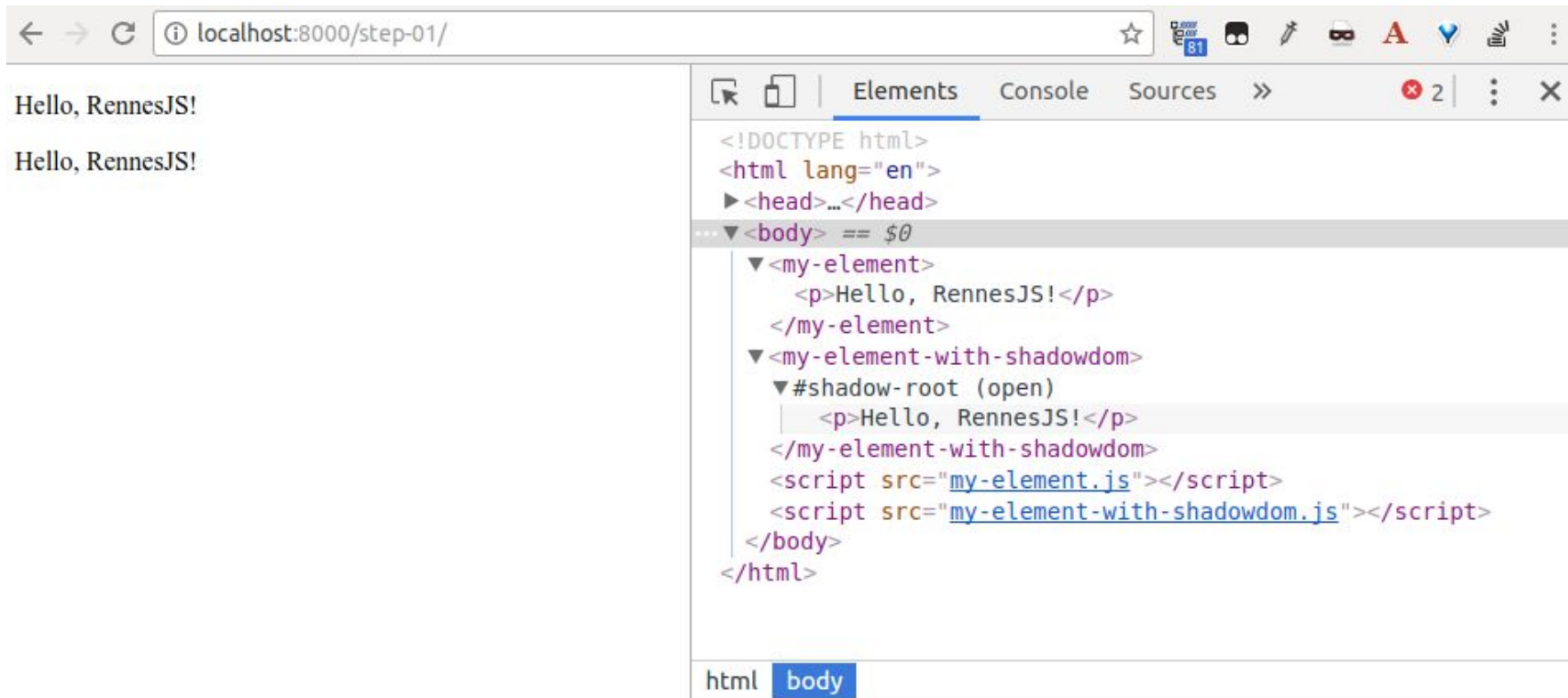
# Adding ShadowDOM

```
class MyElementWithShadowDom extends HTMLElement {  
  
  // This gets called when the HTML parser sees your tag  
  constructor() {  
    super(); // always call super() first in the ctor.  
    this.msg = 'Hello, RennesJS!';  
    this.attachShadow({ mode: 'open' });  
  }  
  // Called when your element is inserted in the DOM or  
  // immediately after the constructor if it's already in the DOM  
  connectedCallback() {  
    this.shadowRoot.innerHTML = `

${this.msg}</p>`;  
  }  
}  
  
customElements.define('my-element-with-shadowdom', MyElementWithShadowDom);


```

# Adding ShadowDOM



localhost:8000/step-01/

Hello, RennesJS!  
Hello, RennesJS!

```
<!DOCTYPE html>  
<html lang="en">  
  <head>...</head>  
  <body> == $0  
    <my-element>  
      <p>Hello, RennesJS!</p>  
    </my-element>  
    <my-element-with-shadowdom>  
      <#shadow-root (open)>  
        <p>Hello, RennesJS!</p>  
      </my-element-with-shadowdom>  
      <script src="my-element.js"></script>  
      <script src="my-element-with-shadowdom.js"></script>  
    </body>  
</html>
```

html body

# Lifecycle callbacks

```

class MyElementLifecycle extends HTMLElement {
  constructor() {
    // Called when an instance of the element is created or upgraded
    super(); // always call super() first in the ctor.
  }
  // Tells the element which attributes to observe for changes
  // This is a feature added by Custom Elements
  static get observedAttributes() {
    return [];
  }
  connectedCallback() {
    // Called every time the element is inserted into the DOM
  }
  disconnectedCallback() {
    // Called every time the element is removed from the DOM.
  }
  attributeChangedCallback(attrName, oldVal, newVal) {
    // Called when an attribute was added, removed, or updated
  }
  adoptedCallback() {
    // Called if the element has been moved into a new document
  }
}

```

# my-counter custom element

```

class MyCounter extends HTMLElement {

  constructor() {
    super();
    this._counter = 0;
    this.attachShadow({ mode: 'open' });
  }

  connectedCallback() { this.render() }
  static get observedAttributes() { return [ 'counter' ] }

  attributeChangedCallback(attr, oldVal, newVal) {
    if (oldVal !== newVal) {
      this[attr] = newVal;
    }
  }
}

```

# my-counter custom element

```

get counter() {
  return this._counter;
}

set counter(value) {
  if (value !== this._counter) {
    this._counter = Number.parseInt(value);
    this.setAttribute('counter', value);
    this.display();
  }
}

increment() {
  this.counter = this.counter + 1;
}

```

# my-counter custom element

```

render() {
  let button = document.createElement('button');
  button.innerHTML = '+';
  button.addEventListener('click', this.increment.bind(this));
  this.shadowRoot.appendChild(button);

  this.output = document.createElement('span');
  this.shadowRoot.appendChild(this.output);

  this.style.display = 'block';
  this.style.fontSize = '5rem';
  button.style.fontSize = '5rem';
  button.style.borderRadius = '1rem';
  button.style.padding = '0.5rem 2rem';
  this.output.style.marginLeft = '2rem';
}

display() {
  this.output.innerHTML = `${this.counter}`;
}

```

# my-counter custom element

+ 42



# Polymer

---

**Adding syntactic sugar to the standard**

# Everything is better with sugar

```

<link rel="import" href="./bower_components/polymer/polymer.html">

<dom-module id="my-polymer-counter">
  <template>
    <style>
      :host {
        font-size: 5rem;
      }
      button {
        font-size: 5rem;
        border-radius: 1rem;
        padding: 0.5rem 2rem;
      }
    </style>
    <button on-click="increment">+</button>
    <span>[[counter]]</span>
  </template>

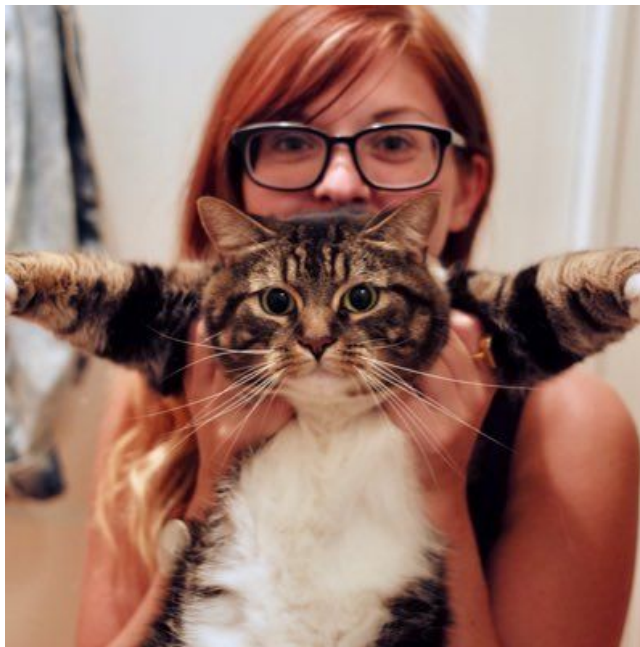
```

# Everything is better with sugar

```
<script>
  class MyPolymerCounter extends Polymer.Element {
    static get is() { return 'my-polymer-counter'; }
    static get properties() {
      return {
        counter: { type: Number, reflectToAttribute:true, value: 0 }
      }
    }
    increment() {
      this.counter = Number.parseInt(this.counter) + 1;
    }
  }

  customElements.define('my-polymer-counter', MyPolymerCounter);
</script>
</dom-module>
```

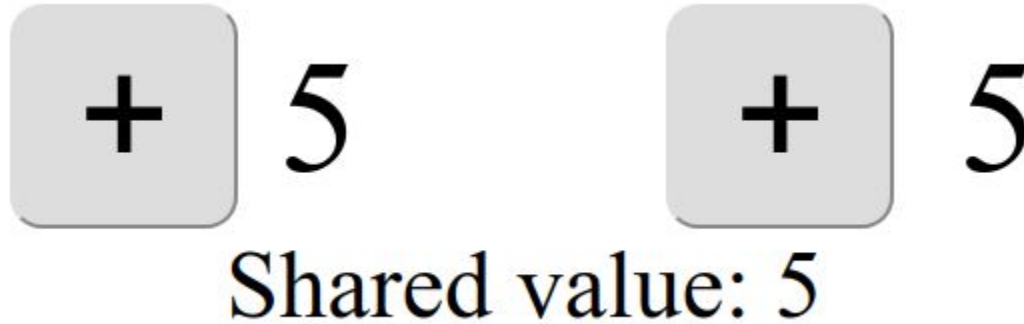
# Everything is better with sugar



*Polymer is like jQuery for Web components*

@notwaldorf

# But they are still custom elements



100% interoperables

# Interoperation pattern

```

<div class="container">
  <my-polymer-counter
    counter="[[value]]"
    on-counter-changed="_onCounterChanged"></my-polymer-counter>
  <my-counter
    counter="[[value]]"
    on-counter-changed="_onCounterChanged"></my-counter>
</div>

```

Attributes for data in  
 Events for data out

# To infinity and beyond!

---

## There is a world outside Polymer

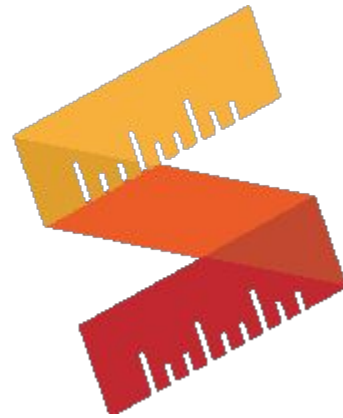
# Lots of web components libraries



For different need and sensibilities

# Slim.js

---





Rapid web components development!

Getting started

Project on Github

Chat on gitter.im

## Introduction

### What is slim.js?

Slim.js is a lightweight web component library that provides extended capabilities for components, such as data binding, using es6 native class inheritance. This library is focused for providing the developer the ability to write robust and native web components without the hassle of dependencies and an overhead of a framework.

# Slim.js

- Lightweight web component library
- Extended capabilities for components
  - data binding
- Using es6 native class inheritance
- Without Shadow DOM

Like a lighter and lesser-featured Polymer

```
Slim.tag('my-slim-counter', `
  <style> [...] </style>
  <div class="container">
    <div class="button" slim-id="button">  </div>
    <div class="value" bind> [[counter]] </div>
  </div>`,
class extends Slim {
  onCreate() {
    if (this.counter == undefined) {
      this.counter = Number.parseInt(this.getAttribute('counter'))||0;
    }
    this.button.onclick = () => {
      this.counter++;
      this.dispatchEvent(new CustomEvent('counter-changed', {detail: {counter: this.counter}}));
    }
  }
});
```

# Bram.js

---





## Bram

A simple 3kB web components library

Home

API

Guides

GitHub

### Install the latest:

```
npm install bram --save
```

Or download a [release](#).

## Examples

### Todo app

HTML

JavaScript

```
<template id="todo-template">
  <form on-submit="addTodo">
    <input type="text" name="todo"
      placeholder="What to do?">
    <button type="submit">Add</button>
  </form>

  <ul>
    <template each="{{todos}}">
      <li>{{item}}</li>
    </template>
  </ul>
</template>

<todo-list></todo-list>
```

# Bram.js

- Lightweight web component library
- Extended capabilities for components
  - data binding
- Using es6 native class inheritance
- With Shadow DOM (optional)

Like a lighter and lesser-featured Polymer, with Shadow DOM

# Bram.js

```
let template=`
<style> [...] </style>
  <div class="container">
    <div class="button" on-click="increase">  </div>
    <div class="value" > {{counter}} </div>
  </div>`;

class MyBramCounter extends Bram(HTMLElement) {
  static get template() {
    let t = document.createElement('template');
    t.innerHTML = template;
    return t;
  }
  static get events() { return ['counter-changed']; }
  constructor() {
    super();
    this.model.counter = this.getAttribute('counter') || 0;
  }
  static get observedProperties() { return [ 'counter' ] } //Non documented
  increase() {
    this.model.counter++;
    this.dispatchEvent(new CustomEvent('counter-changed', {detail: {counter: this.model.counter}}));
  }
}
```

# Skatejs

---



README.md

## Skate

commitizen friendly semantic-release

Skate is a library built on top of the [W3C web component specs](#) that enables you to write functional and performant web components with a very small footprint.

- Functional rendering pipeline backed by Google's [Incremental DOM](#).
- Inherently cross-framework compatible. For example, it works seamlessly with - and complements - React and other frameworks.
- It's very fast.
- It works with multiple versions of itself on the page.

HTML

```
<x-hello name="Bob"></x-hello>
```

JavaScript

# Skatejs

- Lightweight web component library
- Functional rendering pipeline
  - backed by Google's Incremental DOM
- Very very fast
- With a React/Preact flavour

Nice if you dislike declarative syntax and DOM...

# Skatejs

```
class MySkateCounter extends skate.Component {
  static get props () {
    return {
      counter: skate.prop.number({ attribute: true })
    };
  }
  renderCallback () {
    console.log("render", skate.h('div', {}, 'hello'));
    return [
      skate.h('style', {}, '.container { [...] }'),
      skate.h('style', '.button { [...] }'),
      skate.h('style', {}, '.value { margin: 0.5rem; color: #eee; }'),
      skate.h('div', { 'class': 'container'},
        skate.h('div', {
          'class': 'button' ,
          'onClick': () => {
            this.counter++;
            skate.emit(this, 'counter-changed', { detail: { counter: this.counter } });
          }
        }),
        skate.h('img', { 'src' : './img/skate.png' })),
      skate.h('div', { 'class': 'value'}, this.counter)
    ];
  }
}
```

# Conclusion

---

That's all folks!



