Türk Telekom Ventures

SIMPLY.JS

stable release v0.8 License MIT Gzipped 20KB Commits 8/month

Simply.js is a web-component library for developing user interfaces. It is created by a designer to make it easy to develop atomic design system components and compositions of design systems. It uses native Custom Elements API of the Web Components standard and provides a single file component concept which helps to write HTML, CSS and JavaScript codes in one encapsulated single file per component.

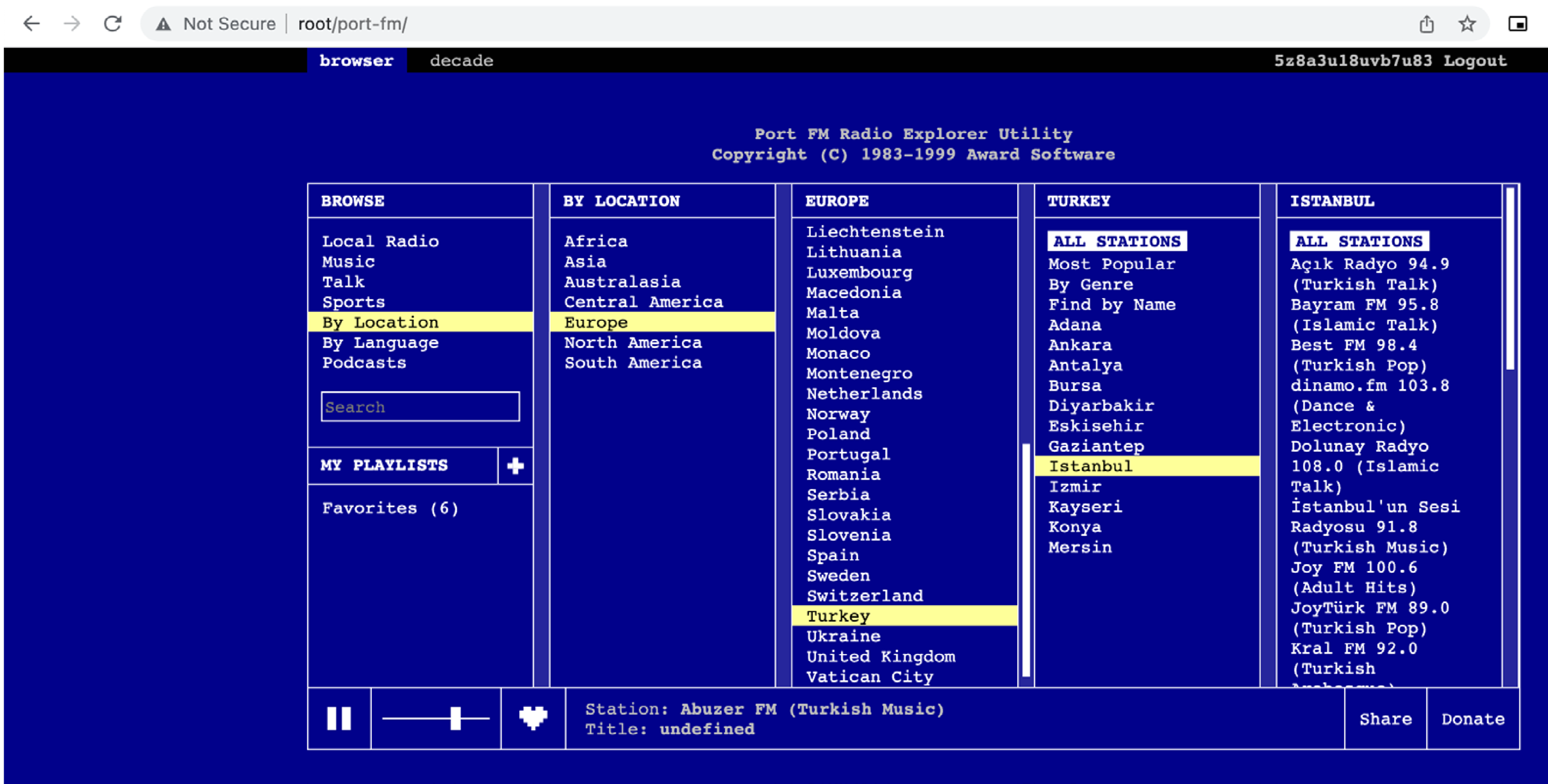
Features

Lighting fast template engine
Reactive DOM
Single file components
Inline components
State management
Tailwind support
Electron support
Router
Zero dependency
No compiler, bundler or builder
Lightweight (20 KB gzipped)
Edge, Chrome, Firefox, Safari, Opera support

SIMPLY.JS

Proof Of Concept

This is an UI of a web based app for discovering online radio stations developed with Simply JS. Design tokens are inspired from old PC Bios. The screenshot taken directly from Chrome.


Not Secure | root/port-fm/

browser decade 5z8a3u18uvb7u83 Logout

Port FM Radio Explorer Utility
Copyright (C) 1983-1999 Award Software

BROWSE	BY LOCATION	EUROPE	TURKEY	ISTANBUL
Local Radio	Africa	Liechtenstein	ALL STATIONS	ALL STATIONS
Music	Asia	Lithuania	Most Popular	Açık Radyo 94.9 (Turkish Talk)
Talk	Australasia	Luxembourg	By Genre	Bayram FM 95.8 (Islamic Talk)
Sports	Central America	Macedonia	Find by Name	Best FM 98.4 (Turkish Pop)
By Location	Europe	Malta	Adana	dinamo.fm 103.8 (Dance & Electronic)
By Language	North America	Moldova	Ankara	Dolunay Radyo 108.0 (Islamic Talk)
Podcasts	South America	Monaco	Antalya	İstanbul'un Sesi Radyosu 91.8 (Turkish Music)
Search		Montenegro	Bursa	Joy FM 100.6 (Adult Hits)
MY PLAYLISTS		Netherlands	Diyarbakir	JoyTürk FM 89.0 (Turkish Pop)
Favorites (6)		Norway	Eskisehir	Kral FM 92.0 (Turkish
		Poland	Gaziantep	
		Portugal	Istanbul	
		Romania	Izmir	
		Serbia	Kayseri	
		Slovakia	Konya	
		Slovenia	Mersin	
		Spain		
		Sweden		
		Switzerland		
		Turkey		
		Ukraine		
		United Kingdom		
		Vatican City		

Station: Abuzer FM (Turkish Music)
Title: undefined

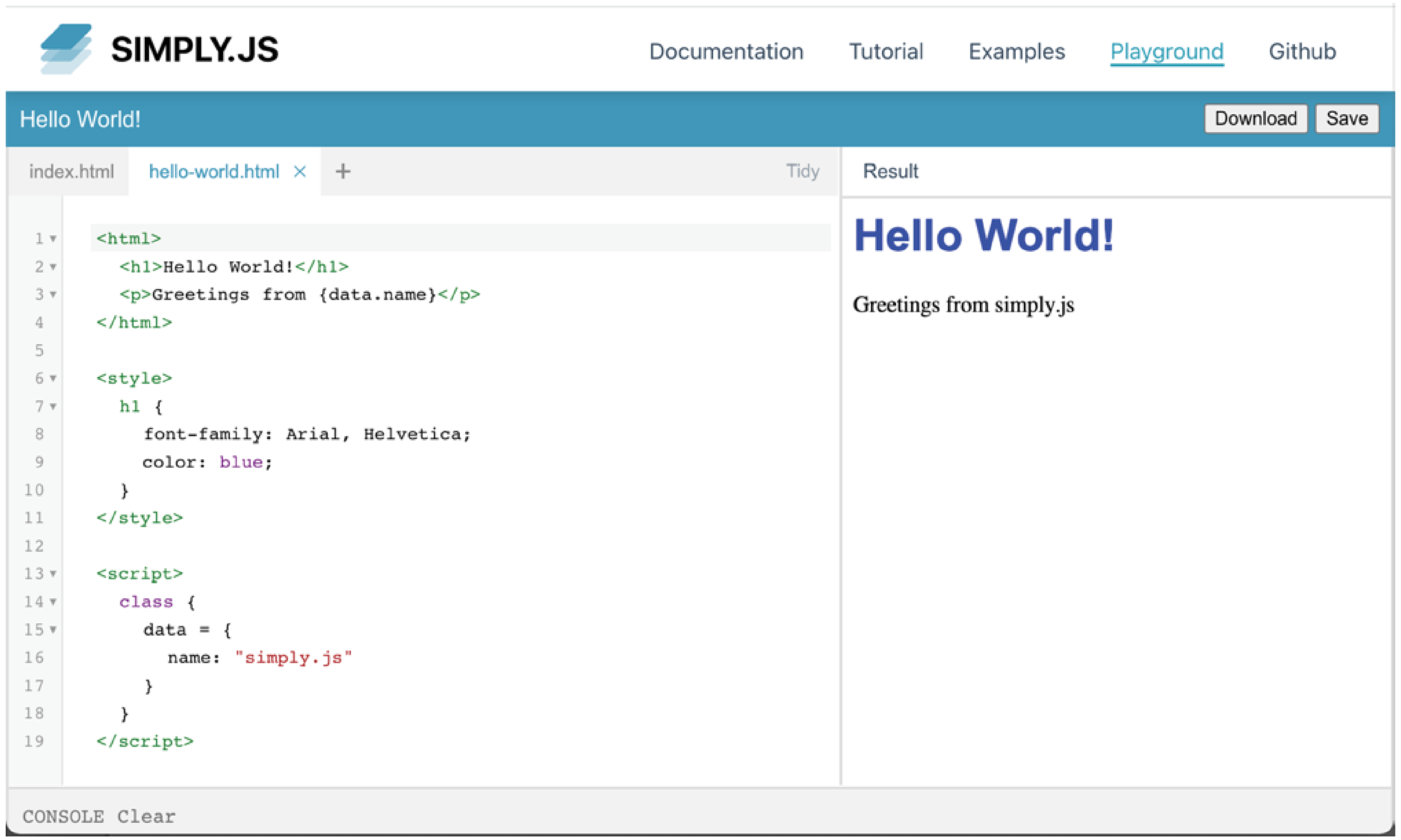
Share Donate

SIMPLY.JS

Playground (REPL)

The REPL of simply.js is at the Playground section of its site and it’s developed with Simply JS too.

It is the fastest way to get start-ed using simply.js. It opens with a "hello world" application and you can freely edit the app anyhow you want. Then you can download the app to con-tinue developing on your local machine.



```
<html>
  <h1>Hello World!</h1>
  <p>Greetings from {data.name}</p>
</html>

<style>
  h1 {
    font-family: Arial, Helvetica;
    color: blue;
  }
</style>

<script>
  class {
    data = {
      name: "simply.js"
    }
  }
</script>
```

Hello World!

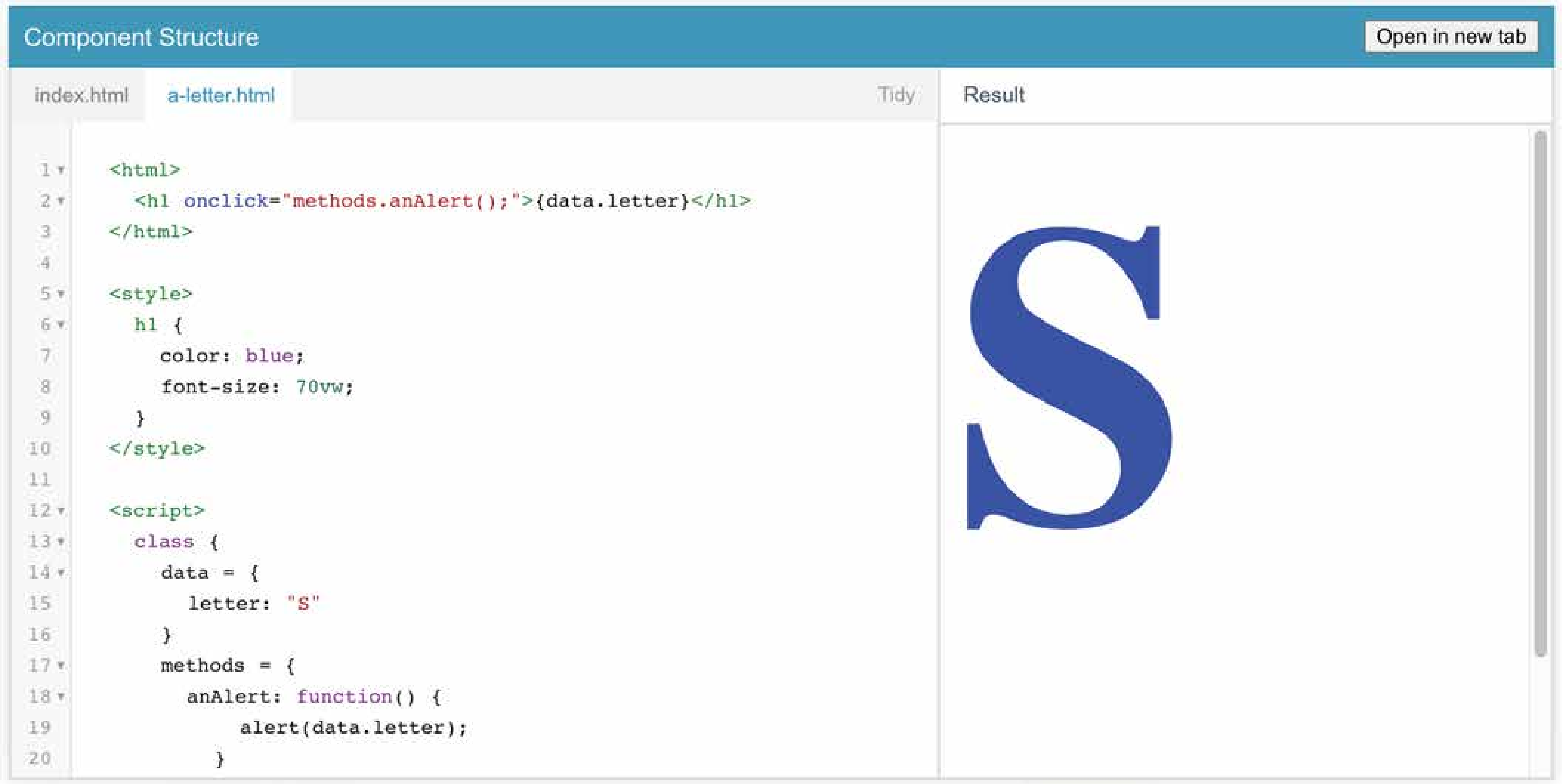
Greetings from simply.js

https://simply.js.org/#/playground

SIMPLY.JS

Component > Structure

Component concept is at the center of simply.js. All other things are shaped around it. Simply.js provide ways for the communication between the components and their orchestration.



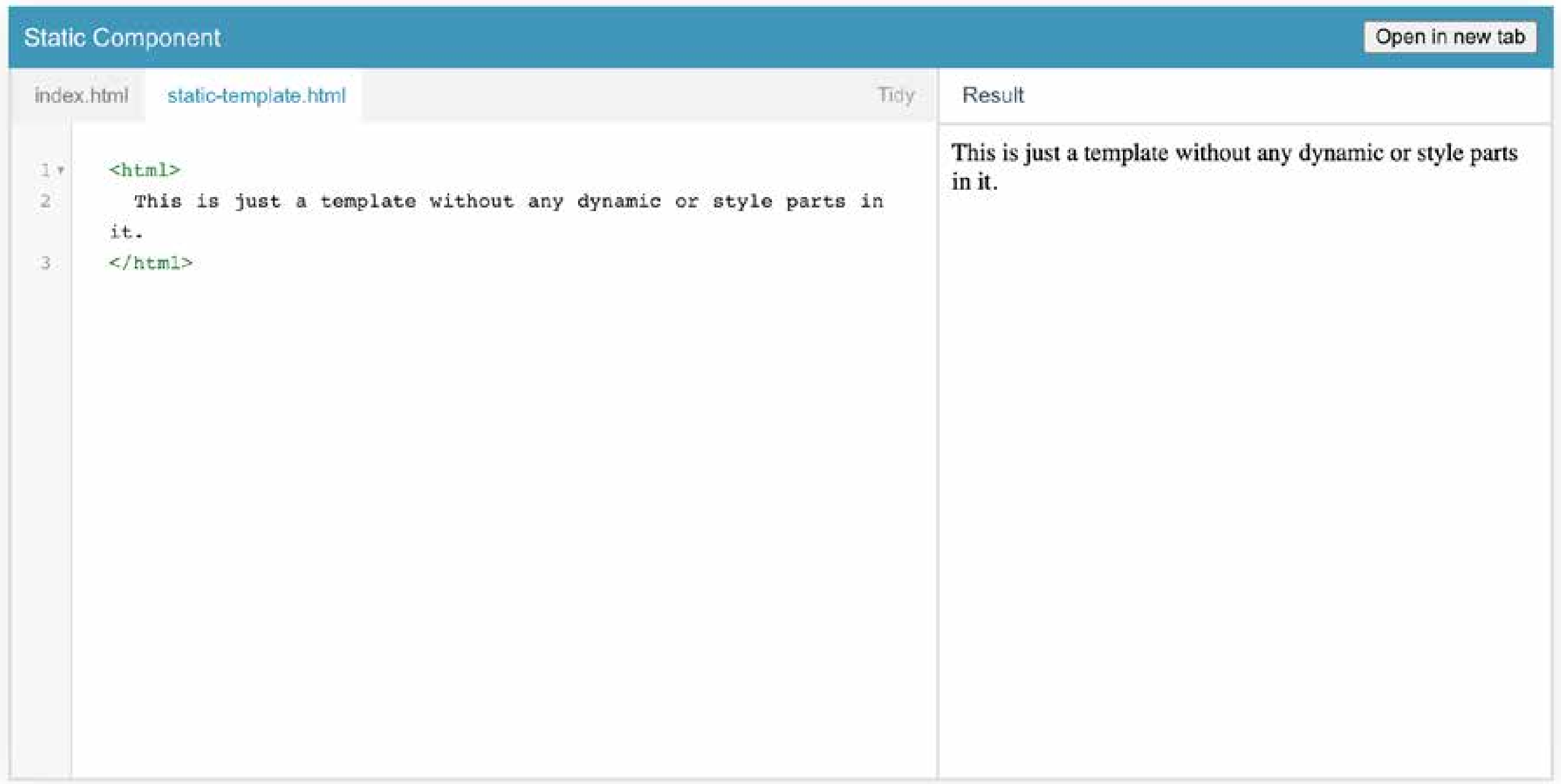
```
<html>
  <h1 onclick="methods.anAlert();">{data.letter}</h1>
</html>

<style>
  h1 {
    color: blue;
    font-size: 70vw;
  }
</style>

<script>
  class {
    data = {
      letter: "S"
    }
    methods = {
      anAlert: function() {
          alert(data.letter);
        }
```

https://simply.js.org/#/playground?to7pgcg4pg47ul1

Component > HTML

Rendering part of the component. The elements in **<html>** tag will be mounted to the DOM after processing by template engine. Almost every component has a **<html>** tag but it is not a rule. Some components only contain **<script>** tag and has some logic in it.

The template engine of simply.js works in **<html>** tag and acts like a superset of HTML. The engine has **conditionals, each loops, reactive variables, literals, expressions, DOM events and nested components.**



```
<html>
  This is just a template without any dynamic or style parts in it.
</html>
```

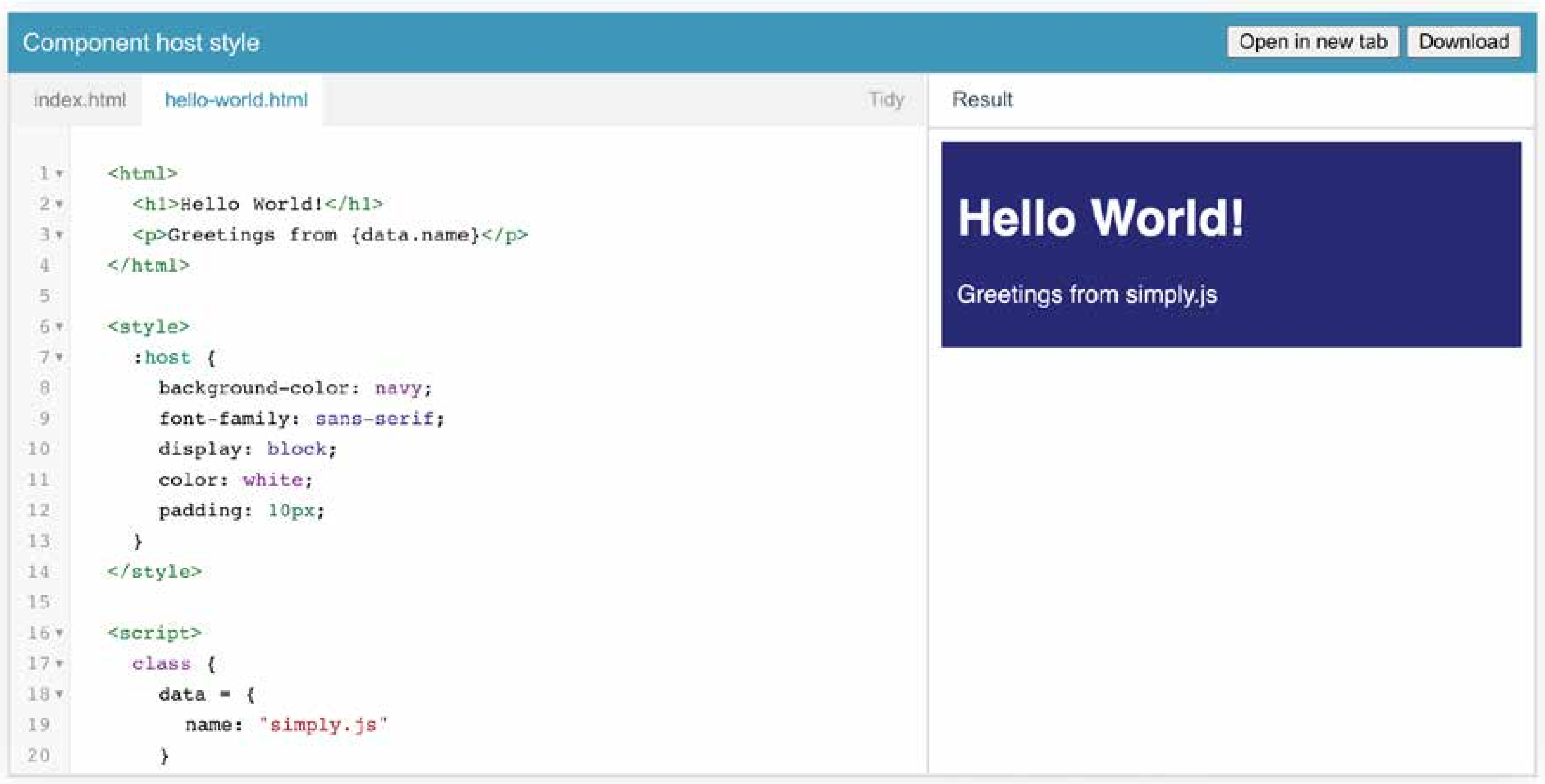
This is just a template without any dynamic or style parts in it.

https://simply.js.org/#/playground?9yl7k6gtgkucjmw

SIMPLY.JS

Component > Style

Encapsulated style definitions only affect the elements inside the template tag of the component. But there is one exception. The inherited styles of the document can affect all child components. For example, when you define color property of body as red in the root document (index.html), then texts of all components inside the document will be red if you don't define otherwise inside the style tag of a component.



```
<html>
  <h1>Hello World!</h1>
  <p>Greetings from {data.name}</p>
</html>

<style>
  :host {
    background-color: navy;
    font-family: sans-serif;
    display: block;
    color: white;
    padding: 10px;
  }
</style>

<script>
  class {
    data = {
      name: "simply.js"
    }
```

Hello World!

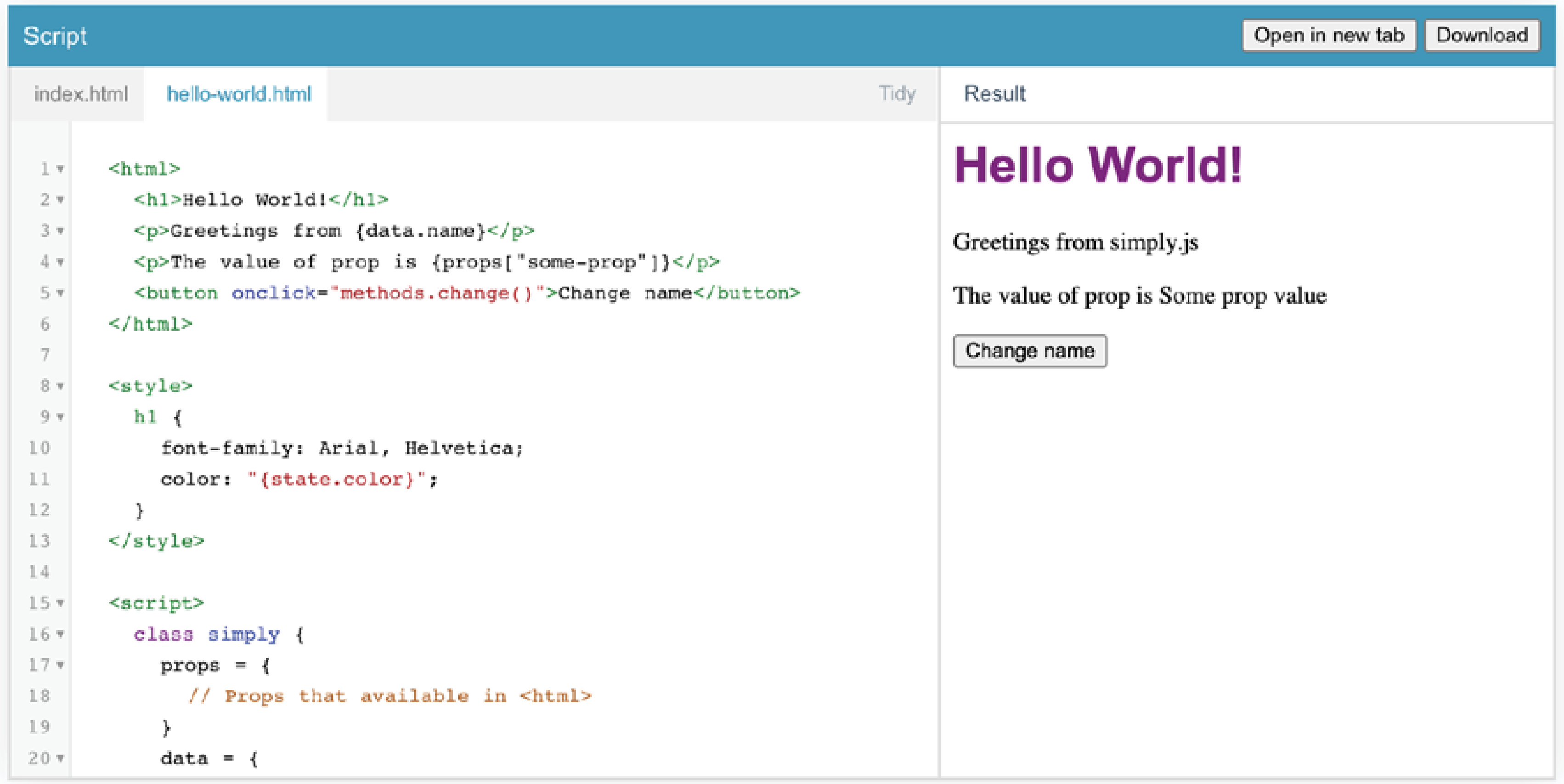
Greetings from simply.js

https://simply.js.org/#/playground?bzr1zokh1i7udmw

SIMPLY.JS

Component > Script

The last and most important part of a component. It contains entire logic and data of a component. It can communi-cate with html and style sections. As you guess, we are writing our script in a **class { ... }** or **class simply { ... }** object. It can hold data, props, lifecycle hooks, variable changes with watch, manage states, and contain methods etc. Here is how an empty component script looks like.



```
<html>
  <h1>Hello World!</h1>
  <p>Greetings from {data.name}</p>
  <p>The value of prop is {props["some-prop"]}</p>
  <button onclick="methods.change()">Change name</button>
</html>

<style>
  h1 {
    font-family: Arial, Helvetica;
    color: "{state.color}";
  }
</style>

<script>
  class simply {
    props = {
      // Props that available in <html>
    }
    data = {
```

Hello World!

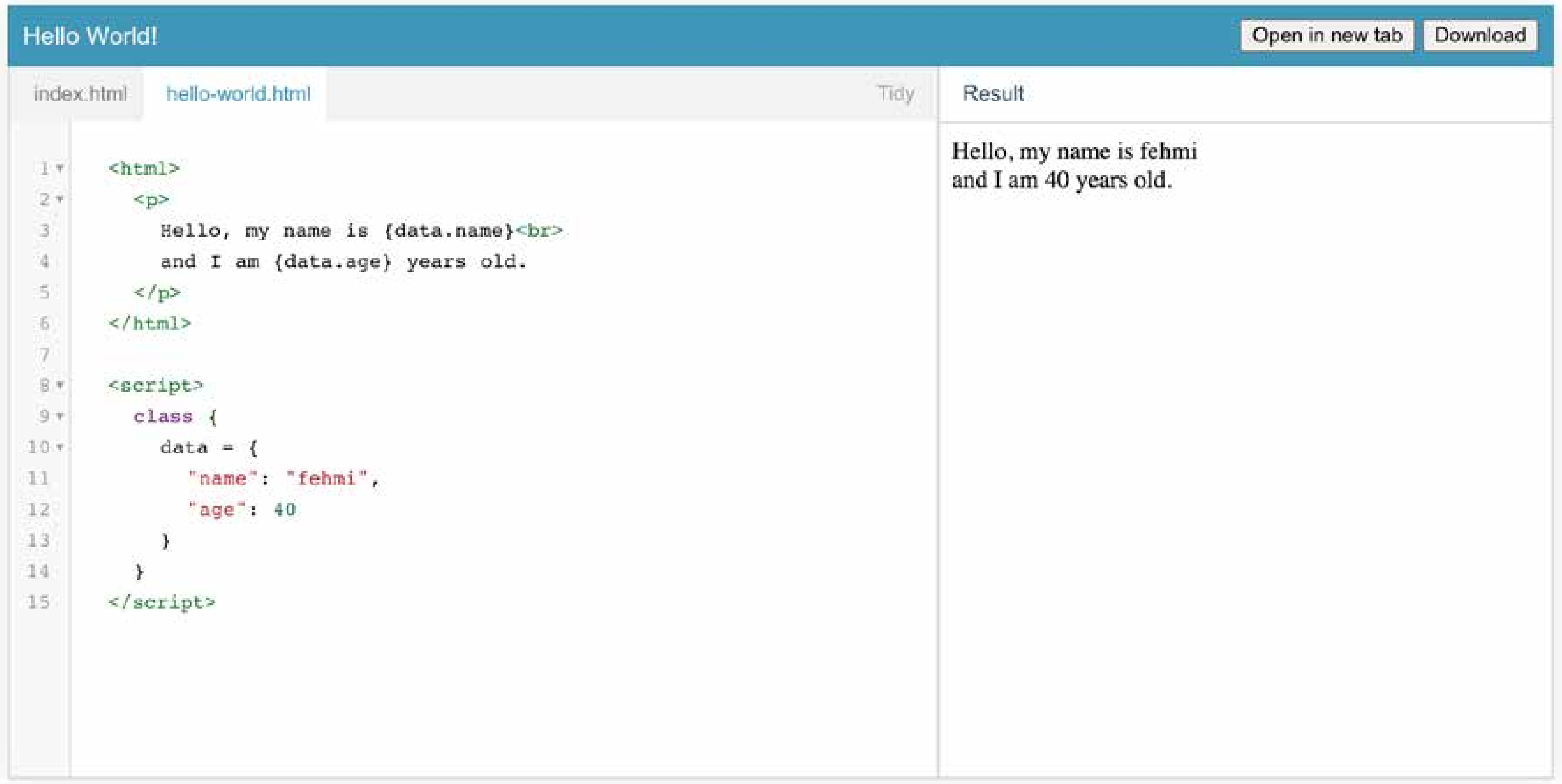
Greetings from simply.js

The value of prop is Some prop value

https://simply.js.org/#/playground?x4x41e8kl1g9n9n

Template Engine > Variables

You can define your variables on the data section in your components. Then you will be able to use them in your template.



```
<html>
  <p>
    Hello, my name is {data.name}<br>
    and I am {data.age} years old.
  </p>
</html>

<script>
  class {
    data = {
      "name": "fehmi",
      "age": 40
    }
  }
</script>
```

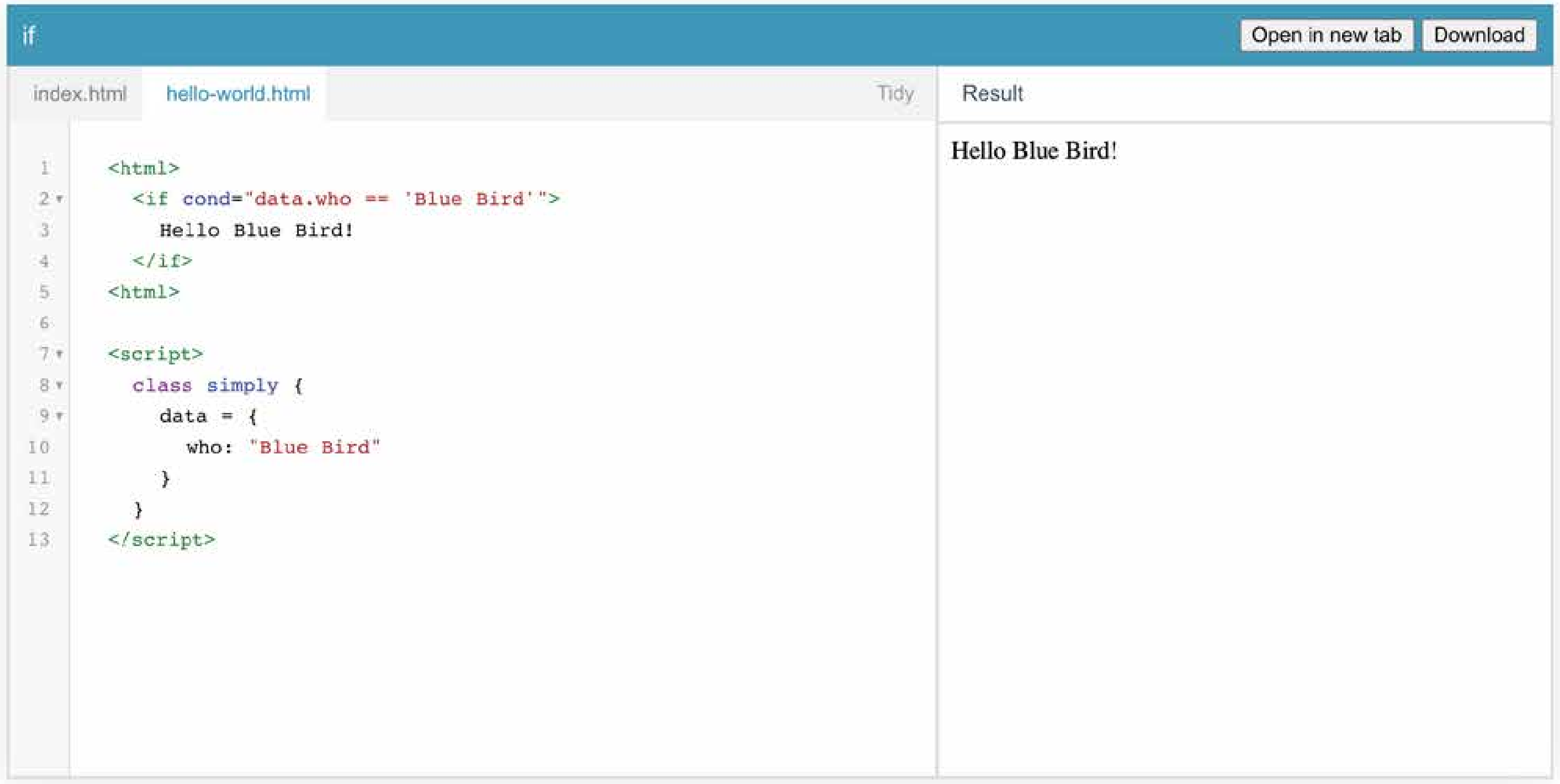
```
Hello, my name is fehmi
and I am 40 years old.
```

https://simply.js.org/#/playground?05loeqiinqkt8pb

SIMPLY.JS

Template Engine > Conditionals

We are defining conditional statemens as special html tags like **<if>**, **<elsif>**, **<else>**. This way helps our IDE to easyly colorize syntax and format the code without and issue.



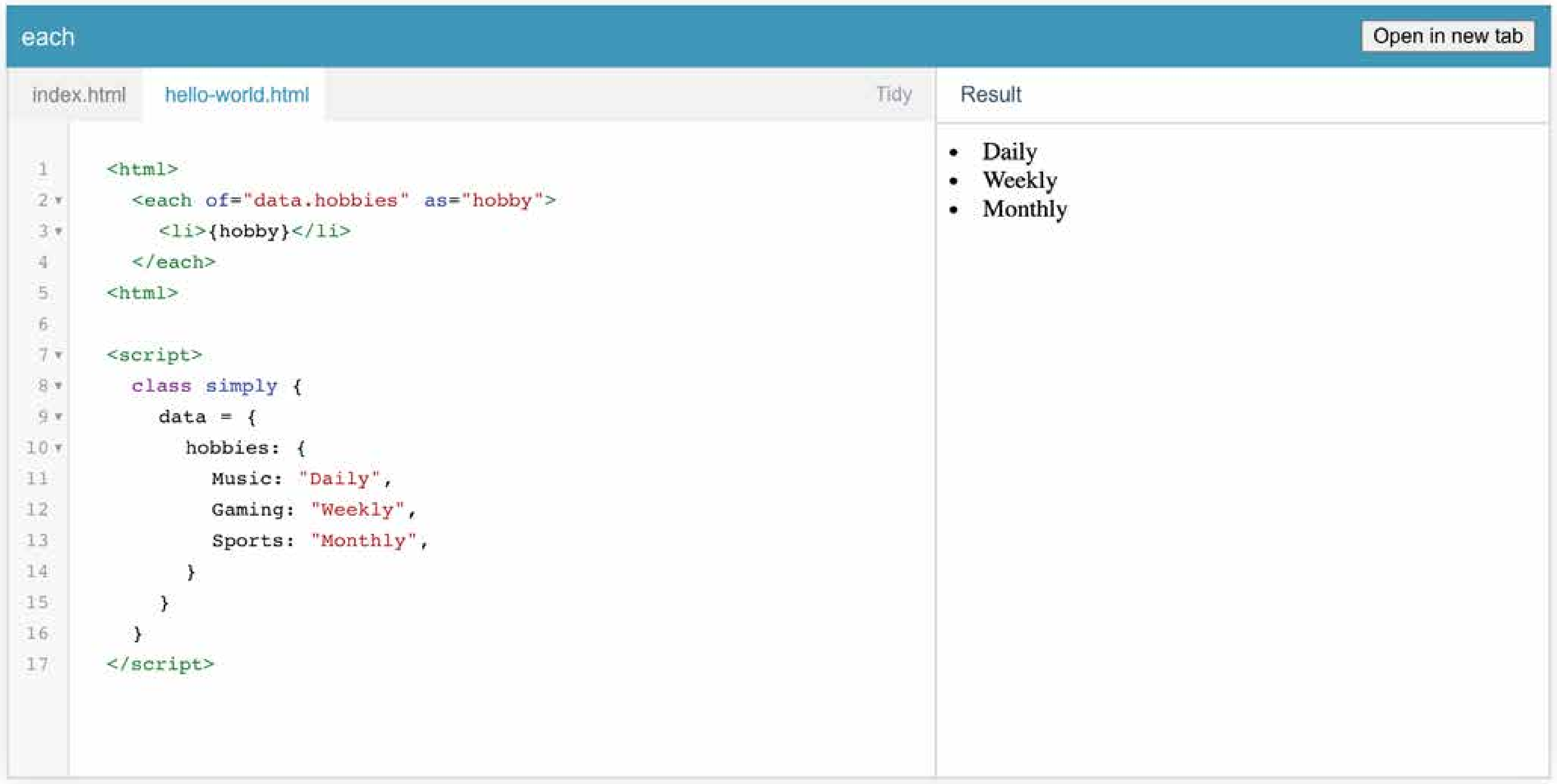
```
<html>
  <if cond="data.who == 'Blue Bird'">
    Hello Blue Bird!
  </if>
<html>

<script>
  class simply {
    data = {
      who: "Blue Bird"
    }
  }
</script>
```

```
Hello Blue Bird!
```

https://simply.js.org/#/playground?th73bi8vvx8q32v

Template Engine > Each Loops

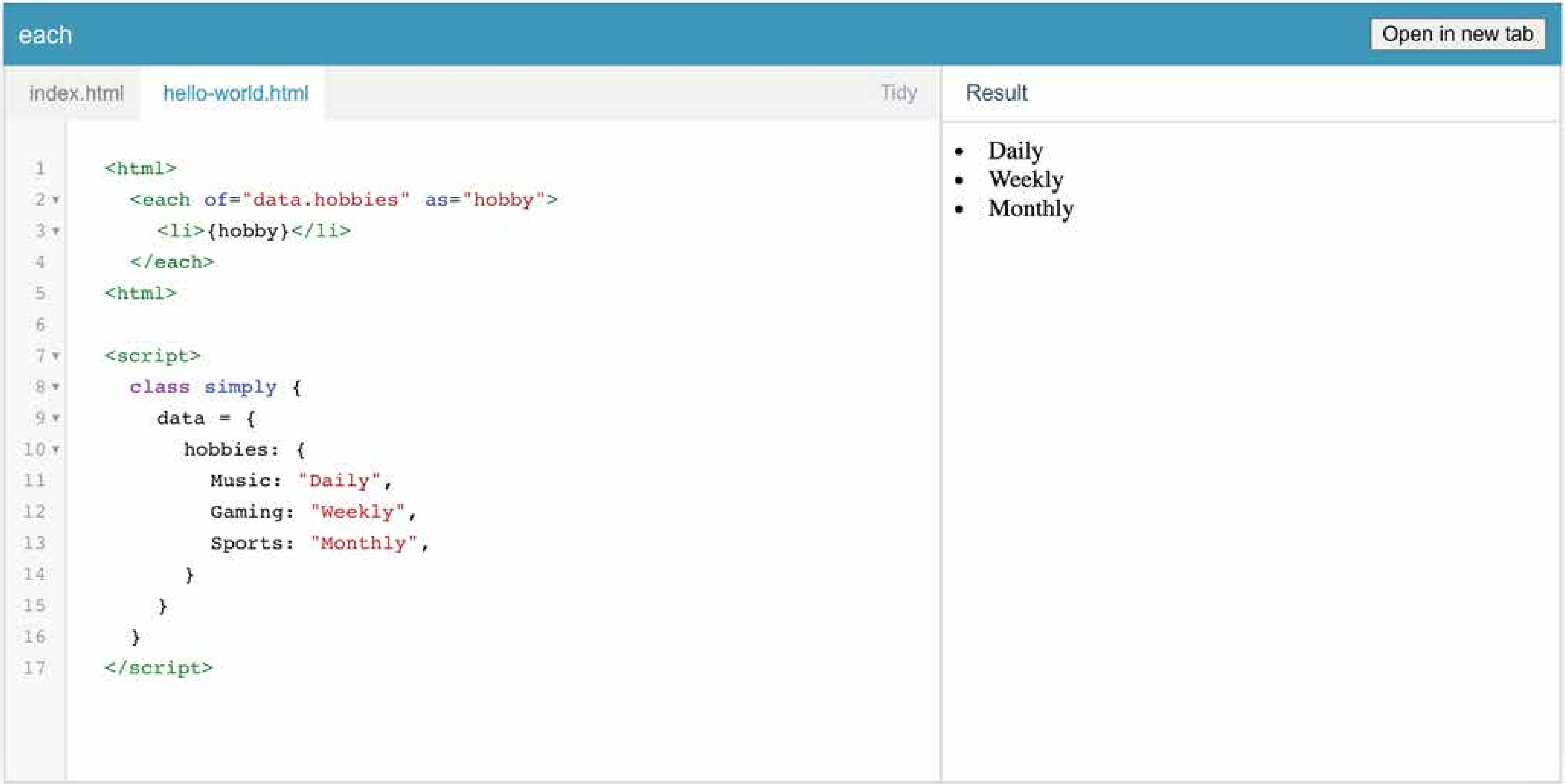
It is possible to walk through with an array or object with simple form of each loops.



```
<html>
  <each of="data.hobbies" as="hobby">
    <li>{hobby}</li>
  </each>
<html>

<script>
  class simply {
    data = {
      hobbies: {
        Music: "Daily",
        Gaming: "Weekly",
        Sports: "Monthly",
      }
    }
  }
</script>
```

- Daily
- Weekly
- Monthly

https://simply.js.org/#/playground?q6idul6gg6jdwby



```
<html>
  <each of="data.hobbies" as="hobby">
    <li>{hobby}</li>
  </each>
<html>

<script>
  class simply {
    data = {
      hobbies: {
        Music: "Daily",
        Gaming: "Weekly",
        Sports: "Monthly",
      }
    }
  }
</script>
```

- Daily
- Weekly
- Monthly

SIMPLY.JS

Template Engine > DOM Events

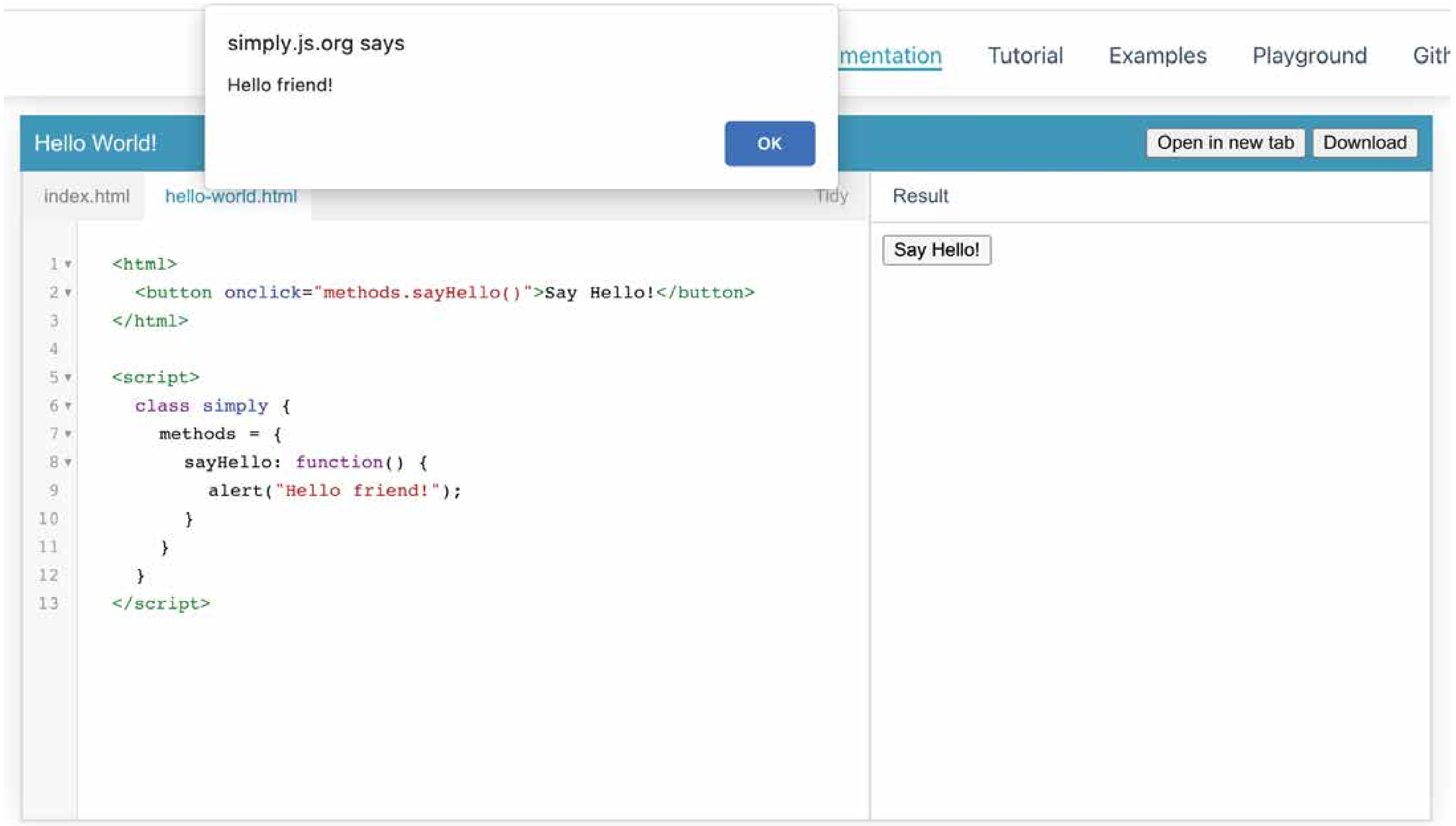
Some of supported events are listed below

onclick
The user clicks an HTML element

onmouseover
The user moves the mouse over an HTML element

onmouseout
The user moves the mouse away from an HTML element

onkeydown
The user pushes a keyboard key



simply.js.org says

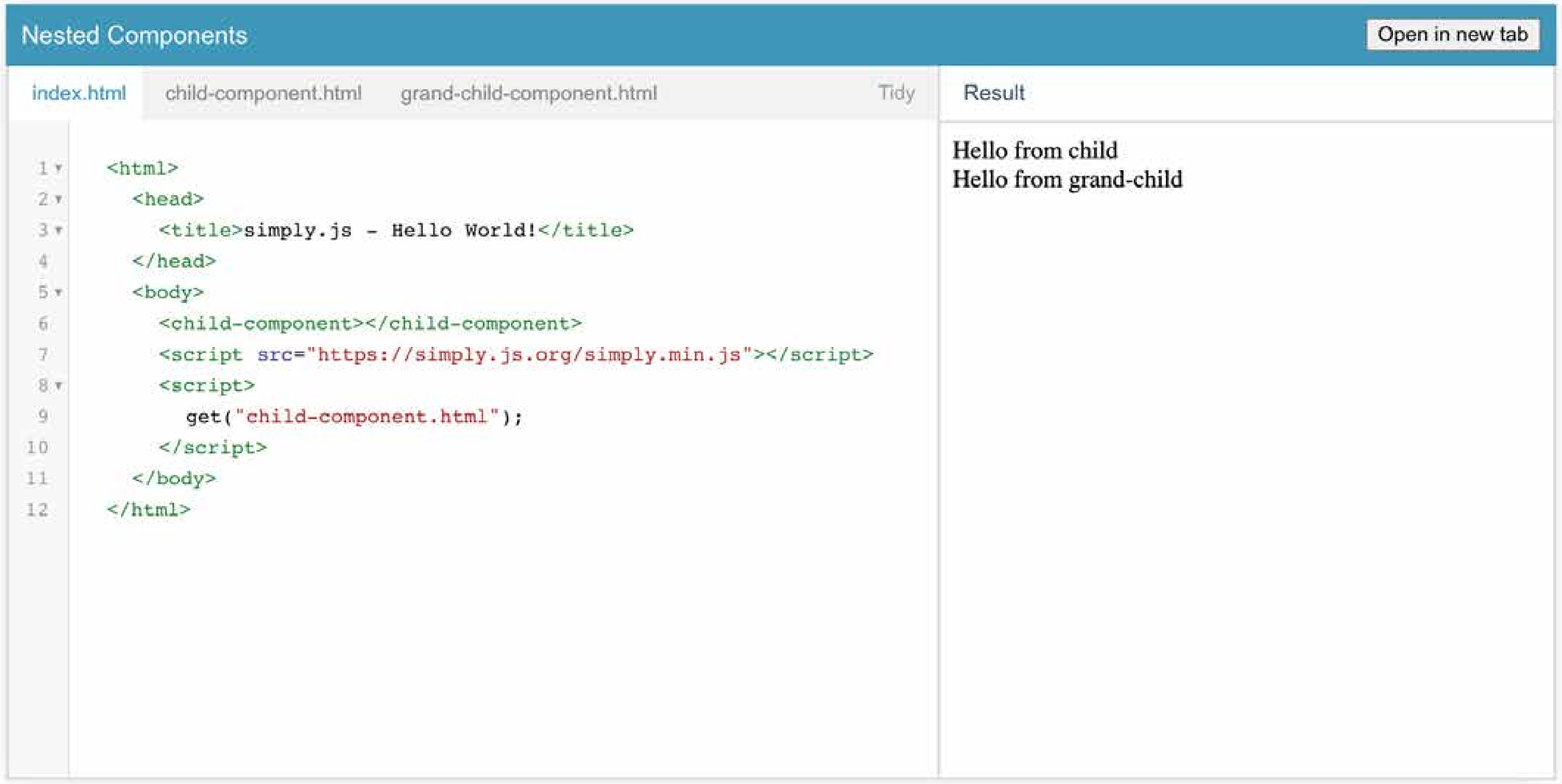
Hello friend!

```
<html>
  <button onclick="methods.sayHello()">Say Hello!</button>
</html>

<script>
  class simply {
    methods = {
      sayHello: function() {
        alert("Hello friend!");
      }
    }
  }
</script>
```

https://simply.js.org/#/playground?q6idul6gg6jdwby

Template Engine > Nested Components

You can call other components inside your components. It has same principle with your main component. You can do it with get function inside at the beginnig of your parent component's script tag. Then you can call it with its tag (child-component) in your template tag. There is no limit about the amount of nested components.



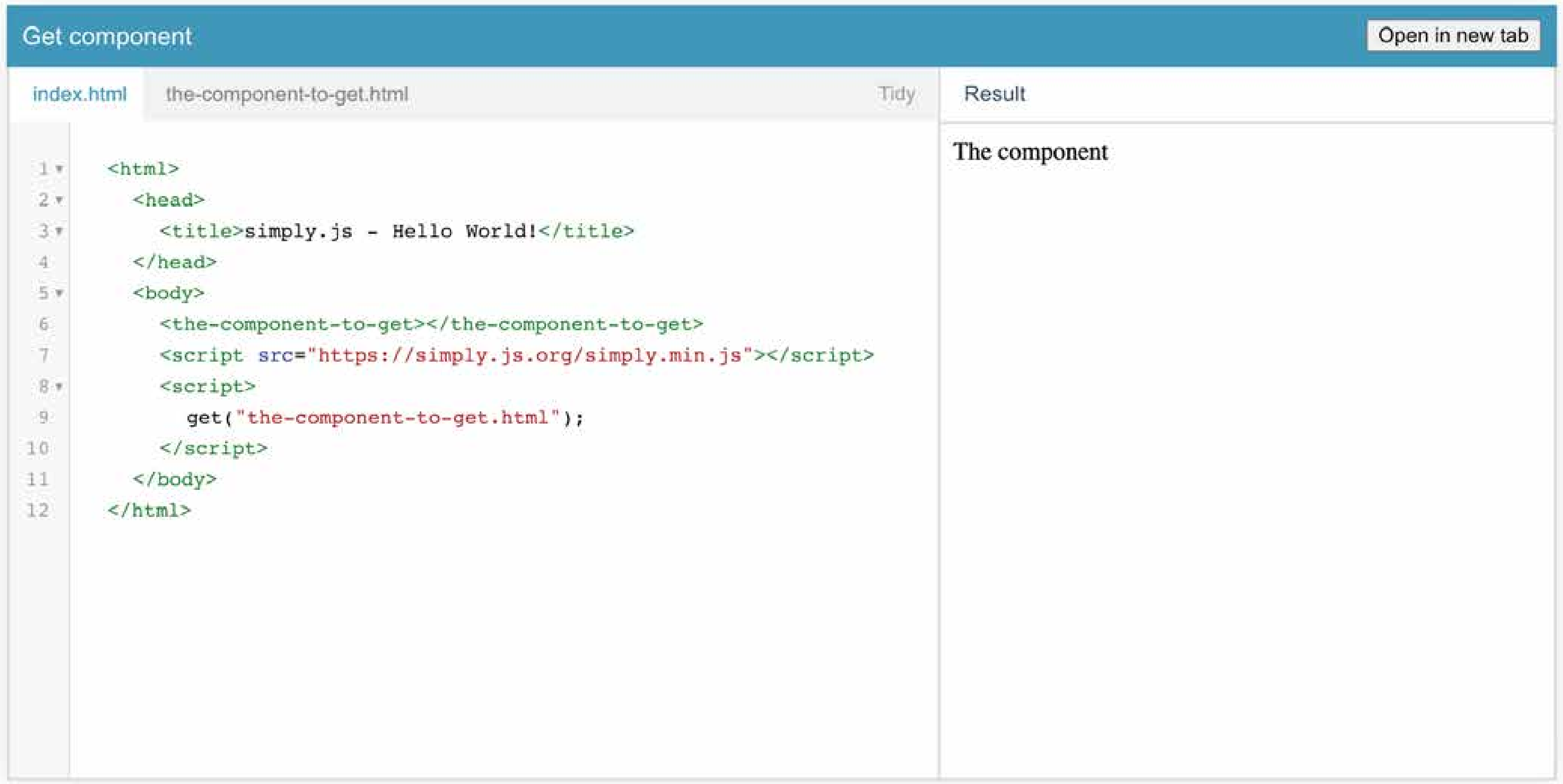
```
<html>
  <head>
    <title>simply.js - Hello World!</title>
  </head>
  <body>
    <child-component></child-component>
    <script src="https://simply.js.org/simply.min.js"></script>
    <script>
      get("child-component.html");
    </script>
  </body>
</html>
```

```
Hello from child
Hello from grand-child
```

https://simply.js.org/#/playground?n7ns3fu6ogtptf2

Script Syntax > Load Components

We load components with the **get()** function. Load components at the top of your component's script area. Filename without extension will be your custom element tag name. This one will be <the-component-to-get> and the content of the file mounted to the element. You can also load multiple components by passing an array to the **get()** function.



```
<html>
  <head>
    <title>simply.js - Hello World!</title>
  </head>
  <body>
    <the-component-to-get></the-component-to-get>
    <script src="https://simply.js.org/simply.min.js"></script>
    <script>
      get("the-component-to-get.html");
    </script>
  </body>
</html>
```

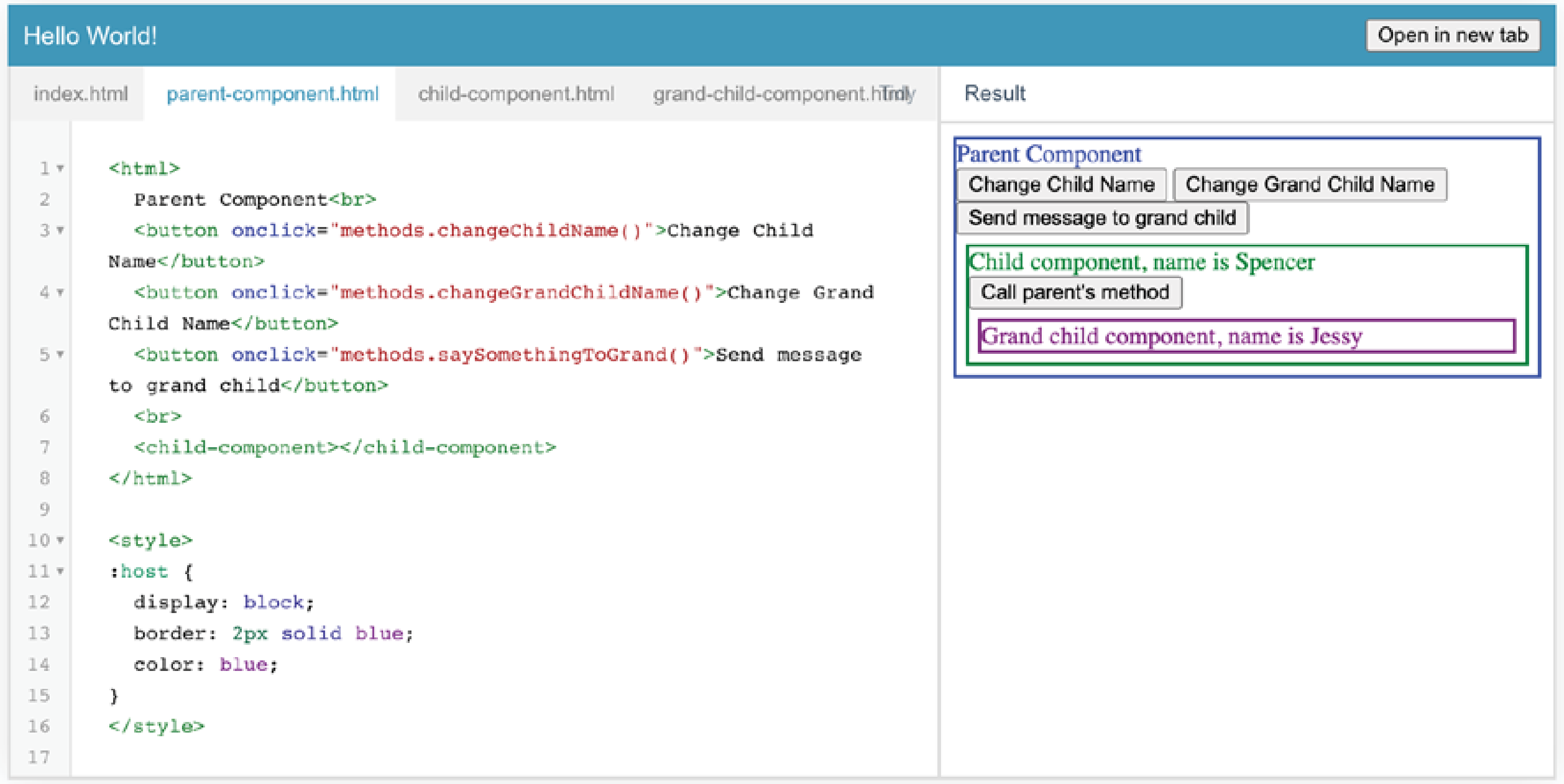
The component

https://simply.js.org/#/playground?2o6tah72m3u0y7v

SIMPLY.JS

Script Syntax > Component Communication

You can directly change a variable in a child component or call a function from it. It is possible to communicate with parent or grand-parent directly to. You can put any kind of values to the attributes. **Object, Array, String, Boolean, Number and even Function** are supported.



```
<html>
  Parent Component<br>
  <button onclick="methods.changeChildName()">Change Child Name</button>
  <button onclick="methods.changeGrandChildName()">Change Grand Child Name</button>
  <button onclick="methods.saySomethingToGrand()">Send message to grand child</button>
  <br>
  <child-component></child-component>
</html>

<style>
:host {
  display: block;
  border: 2px solid blue;
  color: blue;
}
</style>
```

https://simply.js.org/#/playground?4g6y3ikfl9ihlgb

```
// Change a variable in child's data
component.dom.querySelector("child-component").data.name = "New name";

// Call a method from a child
component.dom.querySelector("child-component").methods.functionName();

// Change a variable in grand child's data
var child = component.dom.querySelector("child-component");
child.querySelector("grand-child").data.name = "New name";

// Call a method from a grand child
var child = component.dom.querySelector("child-component");
cchild.querySelector("grand-child").methods.functionName();
```

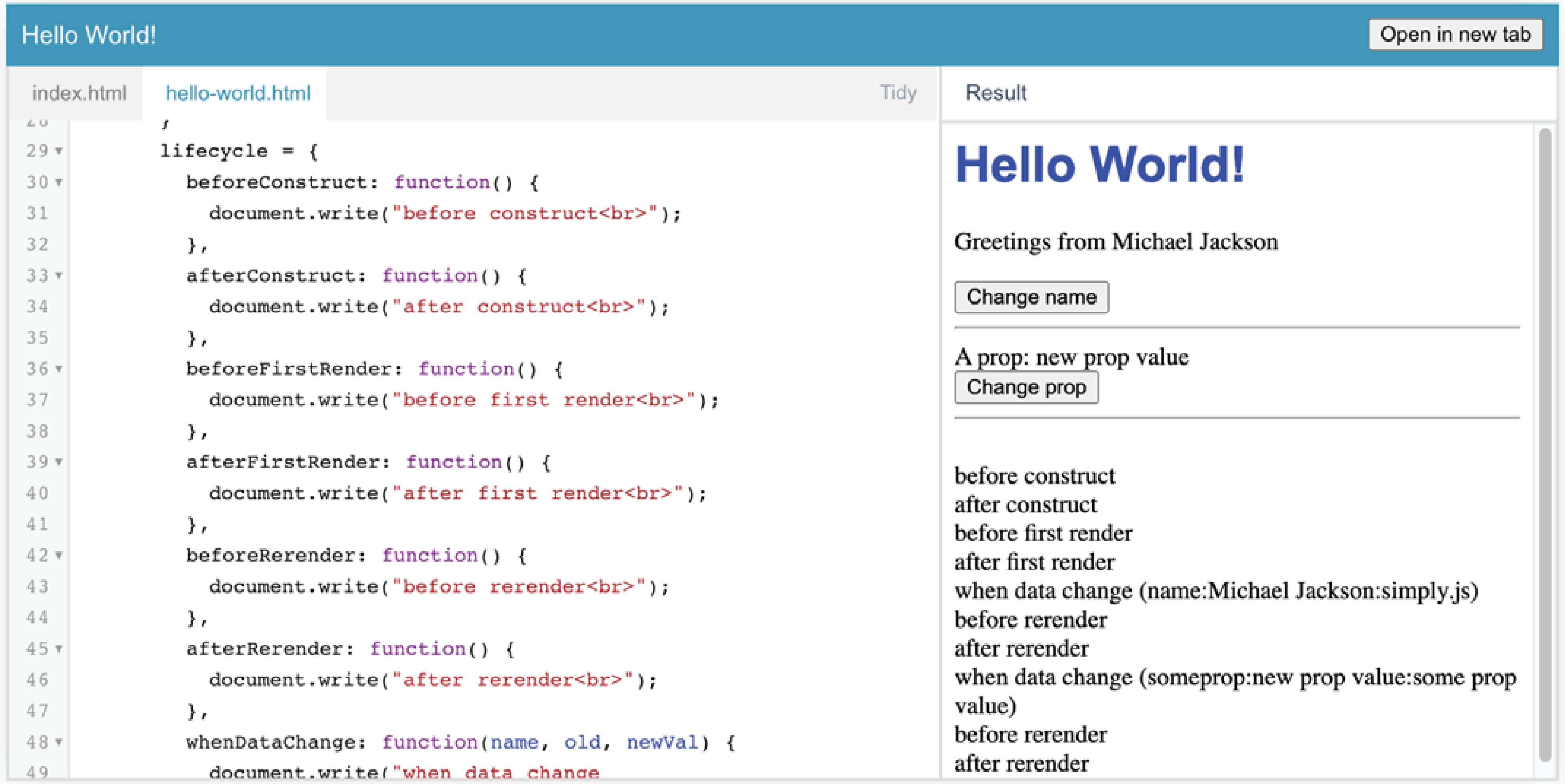
https://simply.js.org/#/docs/component-communication

SIMPLY.JS

Script Syntax > Lifecycle

Lifecycle events are applied to all components and hooks of all phases are designed to control them.

Event	Description
beforeConstruct	Before creating and intializing component
afterConstruct	After creating and intializing component
beforeFirstRender	Before component rendered to the DOM at the first time
afterFirstRender	After component rendered to the DOM at the first time
beforeRerender	Before component rerendered/updated on the DOM
afterRerender	After component rerendered/updated on the DOM
whenDataChange	After a variable in data object of a component is changed
whenPropChange	After a prop is is changed
disconnected	Triggered when the component is removed from the DOM



```
      lifecycle = {
        beforeConstruct: function() {
          document.write("before construct<br>");
        },
        afterConstruct: function() {
          document.write("after construct<br>");
        },
        beforeFirstRender: function() {
          document.write("before first render<br>");
        },
        afterFirstRender: function() {
          document.write("after first render<br>");
        },
        beforeRerender: function() {
          document.write("before rerender<br>");
        },
        afterRerender: function() {
          document.write("after rerender<br>");
        },
        whenDataChange: function(name, old, newVal) {
          document.write("when data change
```

Hello World!

Greetings from Michael Jackson

Change name

A prop: new prop value

Change prop

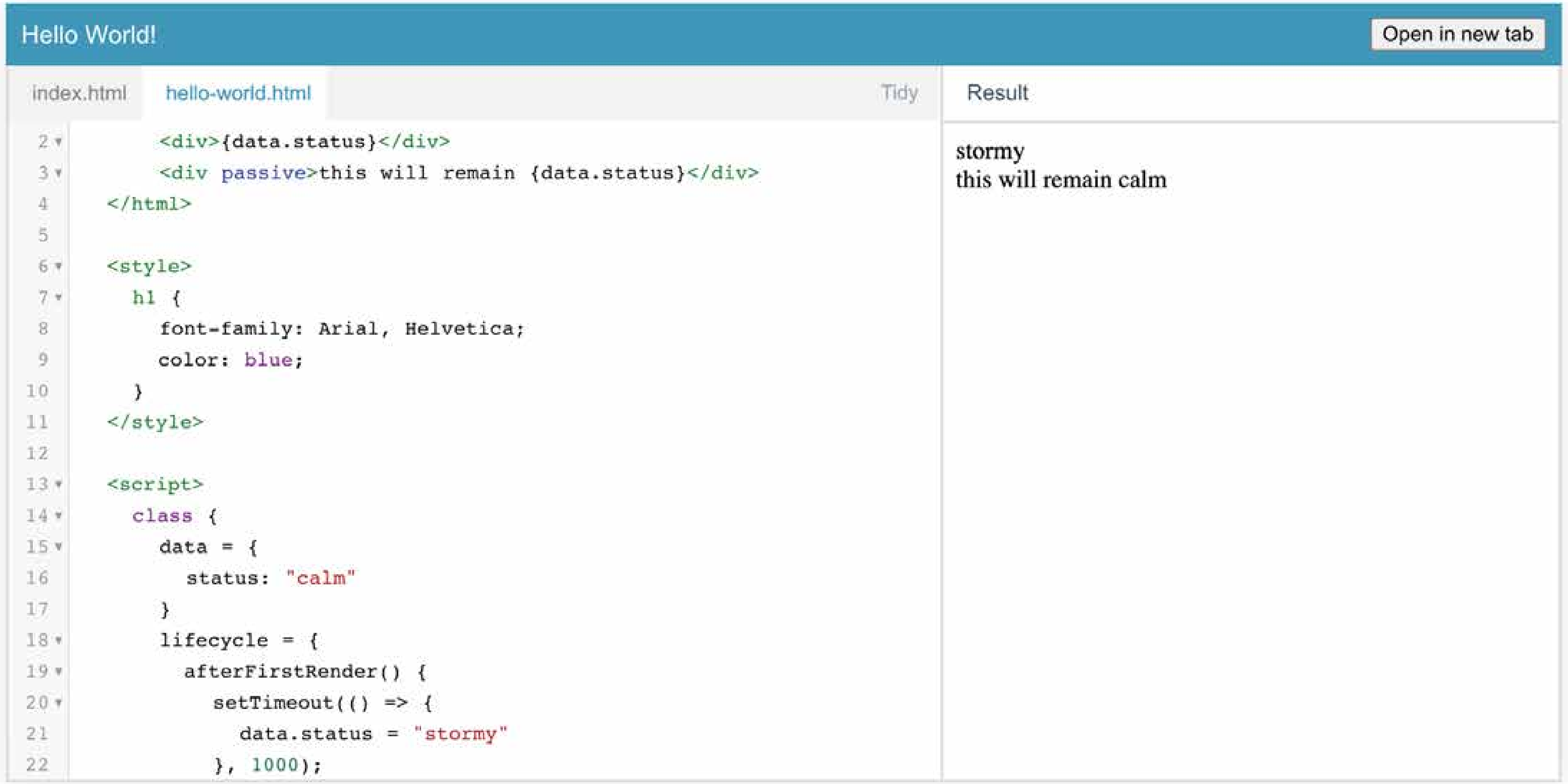
```
before construct
after construct
before first render
after first render
when data change (name:Michael Jackson:simply.js)
before rerender
after rerender
when data change (someprop:new prop value:some prop
value)
before rerender
after rerender
```

https://simply.js.org/#/playground?ugyr2hpkpqeithx

SIMPLY.JS

Script Syntax > Reactivity

All variables you defined on the data section of your com-ponent automaticaly will be reactive. Anytime you change the variable, your template will be rerendered.



```
    <div>{data.status}</div>
    <div passive>this will remain {data.status}</div>
</html>

<style>
  h1 {
    font-family: Arial, Helvetica;
    color: blue;
  }
</style>

<script>
  class {
    data = {
      status: "calm"
    }
    lifecycle = {
      afterFirstRender() {
        setTimeout(() => {
          data.status = "stormy"
        }, 1000);
```

```
stormy
this will remain calm
```

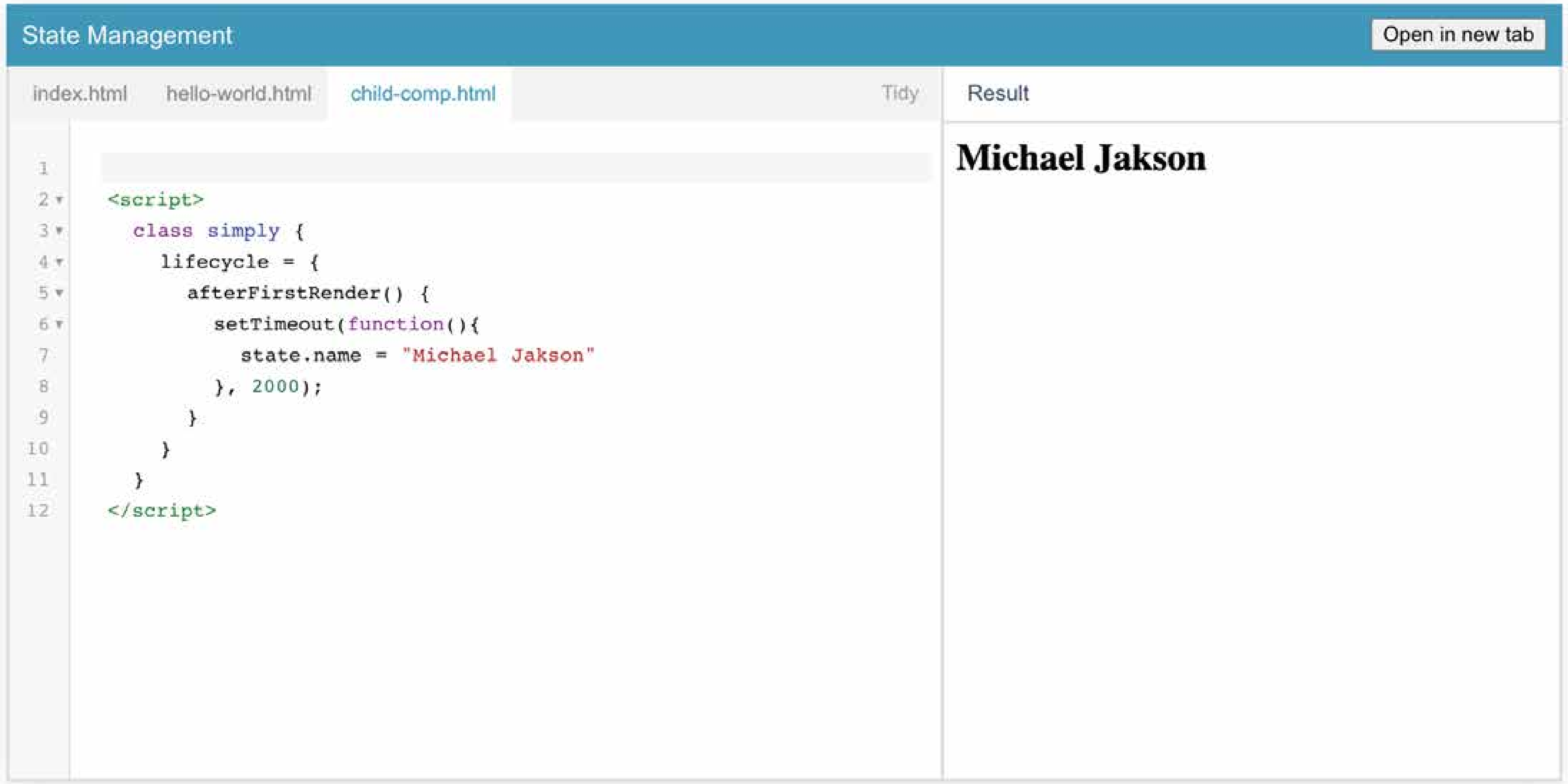
https://simply.js.org/#/playground?k3lkxbulmflvlm9

Script Syntax > State management

State management is pretty easy in simply.js. When you define a state in your component all of child components will share the state and can retrieve or change the state. When a manipulation in the state happen whenever in the component tree all the components that shares the same state will be affected and react to the new value.

The difference between data and state variables is, changes in data only affect the current component, but changes in state variable effects both child and parent components.

Define State in Parent Component and you will be able to access/change from any child component.



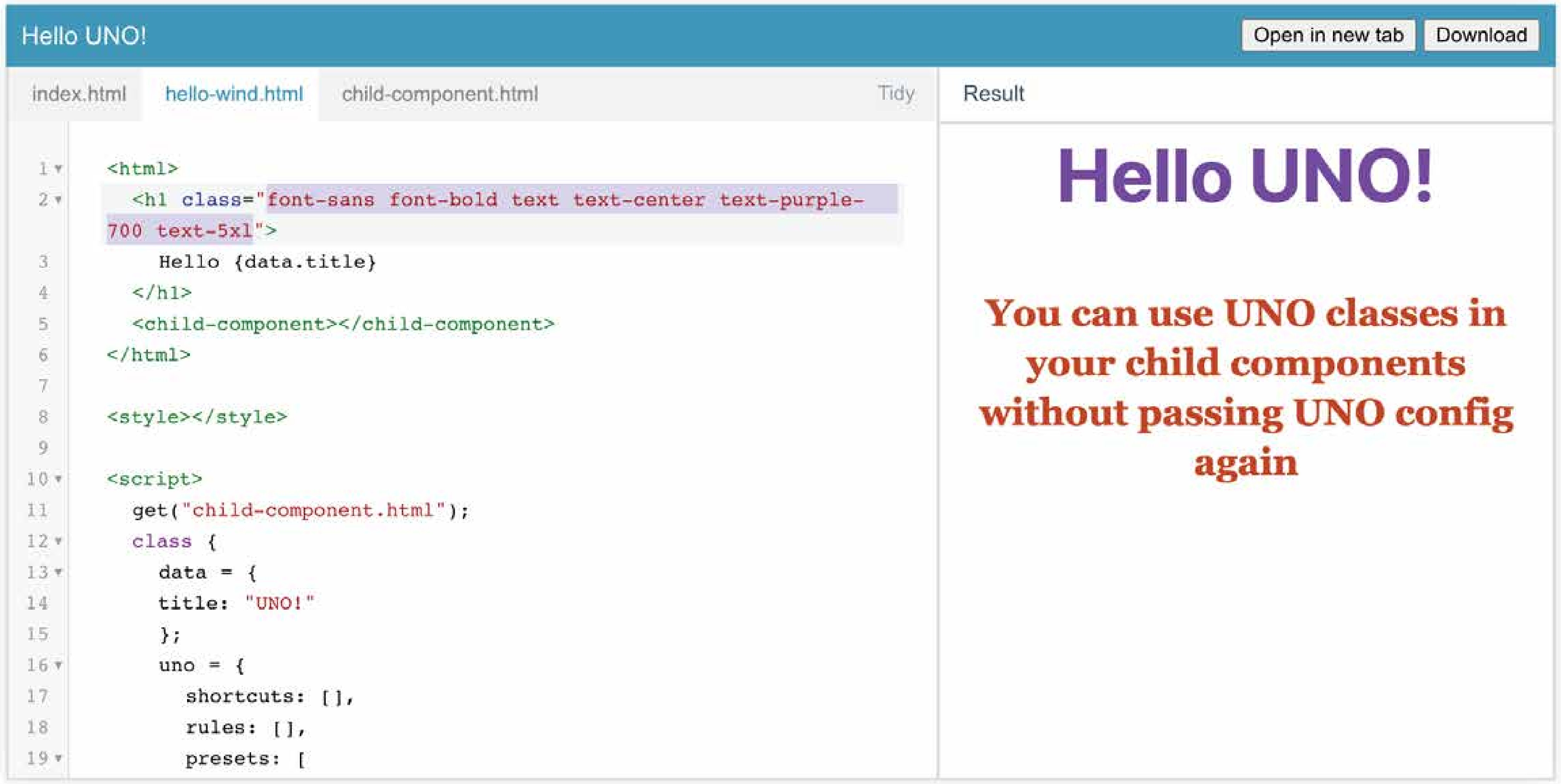
```

<script>
  class simply {
    lifecycle = {
      afterFirstRender() {
        setTimeout(function(){
          state.name = "Michael Jakson"
        }, 2000);
      }
    }
  }
</script>
```

Michael Jakson

https://simply.js.org/#/playground?zpb0cgtjrapmk9s

Style > Tailwind Integration

You can enable Tailwind support (thanks to UNO) for your components and start to use all Tailwind utility classes and even more.



```
<html>
  <h1 class="font-sans font-bold text text-center text-purple-700 text-5xl">
    Hello {data.title}
  </h1>
  <child-component></child-component>
</html>

<style></style>

<script>
  get("child-component.html");
  class {
    data = {
    title: "UNO!"
    };
    uno = {
      shortcuts: [],
      rules: [],
      presets: [
```

Hello UNO!

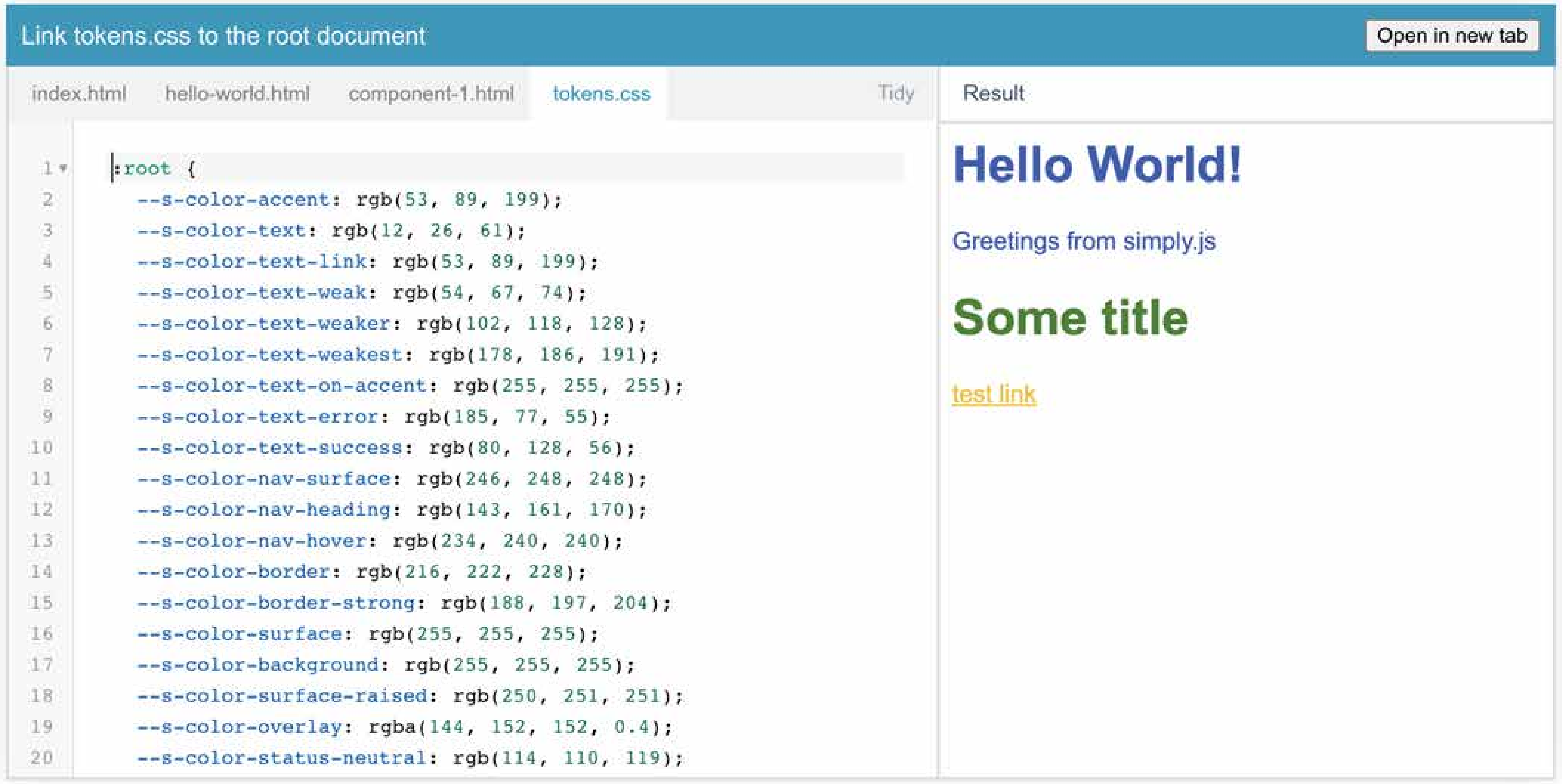
You can use UNO classes in your child components without passing UNO config again

https://simply.js.org/#/playground?ej2yfqzecjv5a0t

SIMPLY.JS

Style > Design Tokens

Simply.js provides several methods for design token management. Just choose one from below that suitable for your needs.



```css
:root {
  --s-color-accent: rgb(53, 89, 199);
  --s-color-text: rgb(12, 26, 61);
  --s-color-text-link: rgb(53, 89, 199);
  --s-color-text-weak: rgb(54, 67, 74);
  --s-color-text-weaker: rgb(102, 118, 128);
  --s-color-text-weakest: rgb(178, 186, 191);
  --s-color-text-on-accent: rgb(255, 255, 255);
  --s-color-text-error: rgb(185, 77, 55);
  --s-color-text-success: rgb(80, 128, 56);
  --s-color-nav-surface: rgb(246, 248, 248);
  --s-color-nav-heading: rgb(143, 161, 170);
  --s-color-nav-hover: rgb(234, 240, 240);
  --s-color-border: rgb(216, 222, 228);
  --s-color-border-strong: rgb(188, 197, 204);
  --s-color-surface: rgb(255, 255, 255);
  --s-color-background: rgb(255, 255, 255);
  --s-color-surface-raised: rgb(250, 251, 251);
  --s-color-overlay: rgba(144, 152, 152, 0.4);
  --s-color-status-neutral: rgb(114, 110, 119);
```

Hello World!

Greetings from simply.js

Some title

test link

https://simply.js.org/#/playground?d6bvgus52kc5f44

SIMPLY.JS

Türk Telekom Ventures

Thanks!

Please visit https://simply.js.org for detailed information

Contact: hello@fozuse.com / 05422664342