What is Vue?

Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is very easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.

Why Vue?

* Extremally lean and small as Its 16 Kbytes Gzipped and minified.
* Great runtime performance.
* It’s very feature reach framework although it’s very small.
* <https://10clouds.com/blog/vuejs-angular-react/>

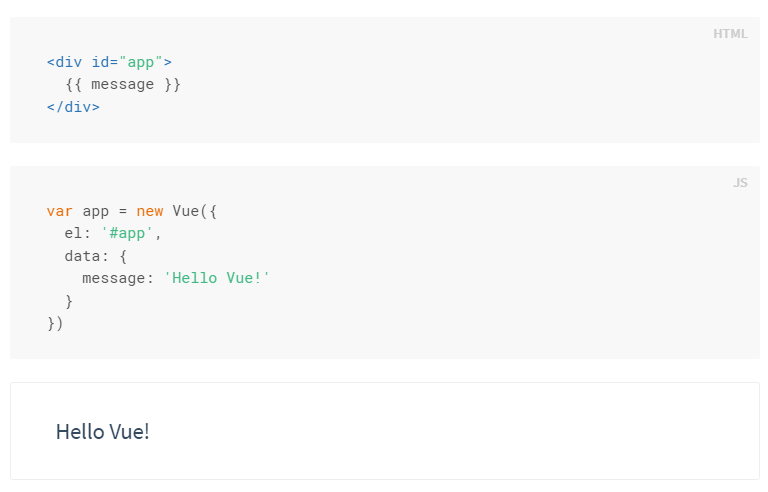
How to install:

Simply you can put a script tag with Vue source inside your html page and the Vue source may be local or from CDN.

After that we will learn advanced environment setup using CLI, WebPack and etc.

Declarative Rendering:

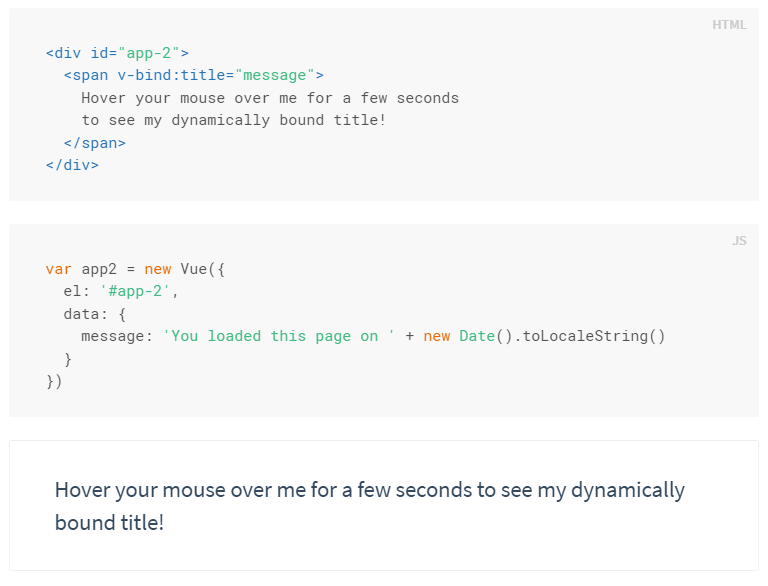
At the core of Vue.js is a system that enables us to declaratively render data to the DOM using straightforward template syntax:



We have already created our very first Vue app! This looks pretty similar to just rendering a string template, but Vue has done a lot of work under the hood. The data and the DOM are now linked, and everything is now **reactive**. How do we know? Just open your browser’s JavaScript console (right now, on this page) and set app.message to a different value. You should see the rendered example above update accordingly.

* el: To define the html area to control.
* data: to define the related variables to bind to.

In addition to text interpolation, we can also bind element attributes like this:



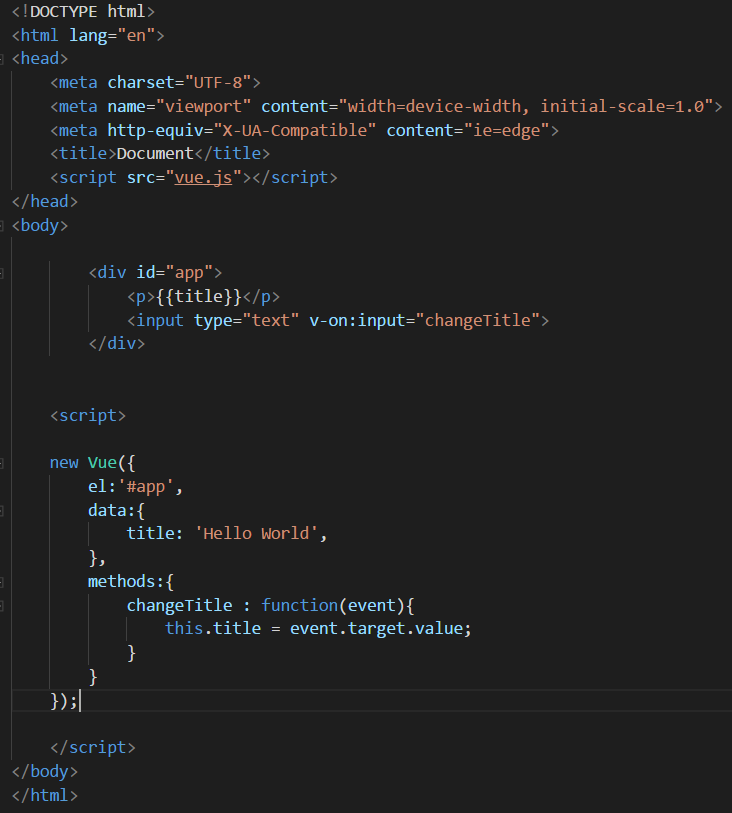
Here we are encountering something new. The v-bind attribute you are seeing is called a **directive**. Directives are prefixed with v- to indicate that they are special attributes provided by Vue, and as you may have guessed, they apply special reactive behavior to the rendered DOM. Here it is basically saying “keep this element’s title attribute up-to-date with the message property on the Vue instance.”

If you open up your JavaScript console again and enter app2.message = 'some new message', you’ll once again see that the bound HTML - in this case the title attribute - has been updated.

Important Libraries and Tutorials:

<https://github.com/vuejs/awesome-vue#components--libraries>

First App Ever:



* As we see in the example we used this instead of data in this.title as Vue by default proxy all the data and methods properties to the Vue object it self so we can access them using this.
* We also here use directive to apply a method when any input typed and we can use v-bind instead as well like in the above example.

The Vue Instance:

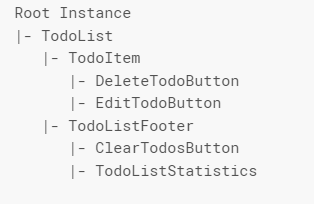
Every Vue application starts by creating a new **Vue instance** with the Vue function:



Although not strictly associated with the [**MVVM pattern**](https://en.wikipedia.org/wiki/Model_View_ViewModel), Vue’s design was partly inspired by it. As a convention, we often use the variable vm (short for ViewModel) to refer to our Vue instance.

When you create a Vue instance, you pass in an **options object**. The majority of this guide describes how you can use these options to create your desired behavior. For reference, you can also browse the full list of options in the [**API reference**](https://vuejs.org/v2/api/#Options-Data).

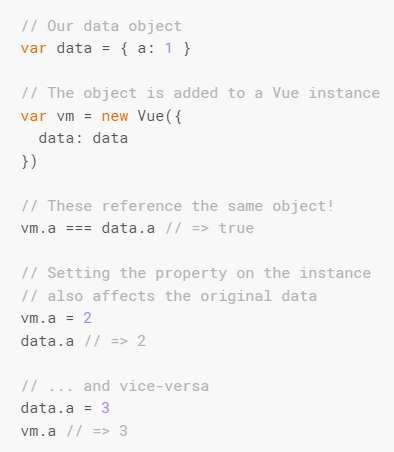
A Vue application consists of a **root Vue instance** created with new Vue, optionally organized into a tree of nested, reusable components. For example, a todo app’s component tree might look like this:



We’ll talk about [**the component system**](https://vuejs.org/v2/guide/components.html) in detail later. For now, just know that all Vue components are also Vue instances, and so accept the same options object (except for a few root-specific options).

Data and Methods:

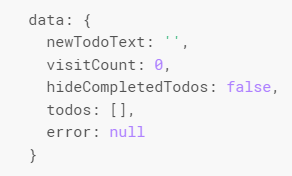
When a Vue instance is created, it adds all the properties found in its data object to Vue’s **reactivity system**. When the values of those properties change, the view will “react”, updating to match the new values.



When this data changes, the view will re-render. It should be noted that properties in data are only **reactive** if they existed when the instance was created. That means if you add a new property, like:



Then changes to b will not trigger any view updates. If you know you’ll need a property later, but it starts out empty or non-existent, you’ll just need to set some initial value. For example:

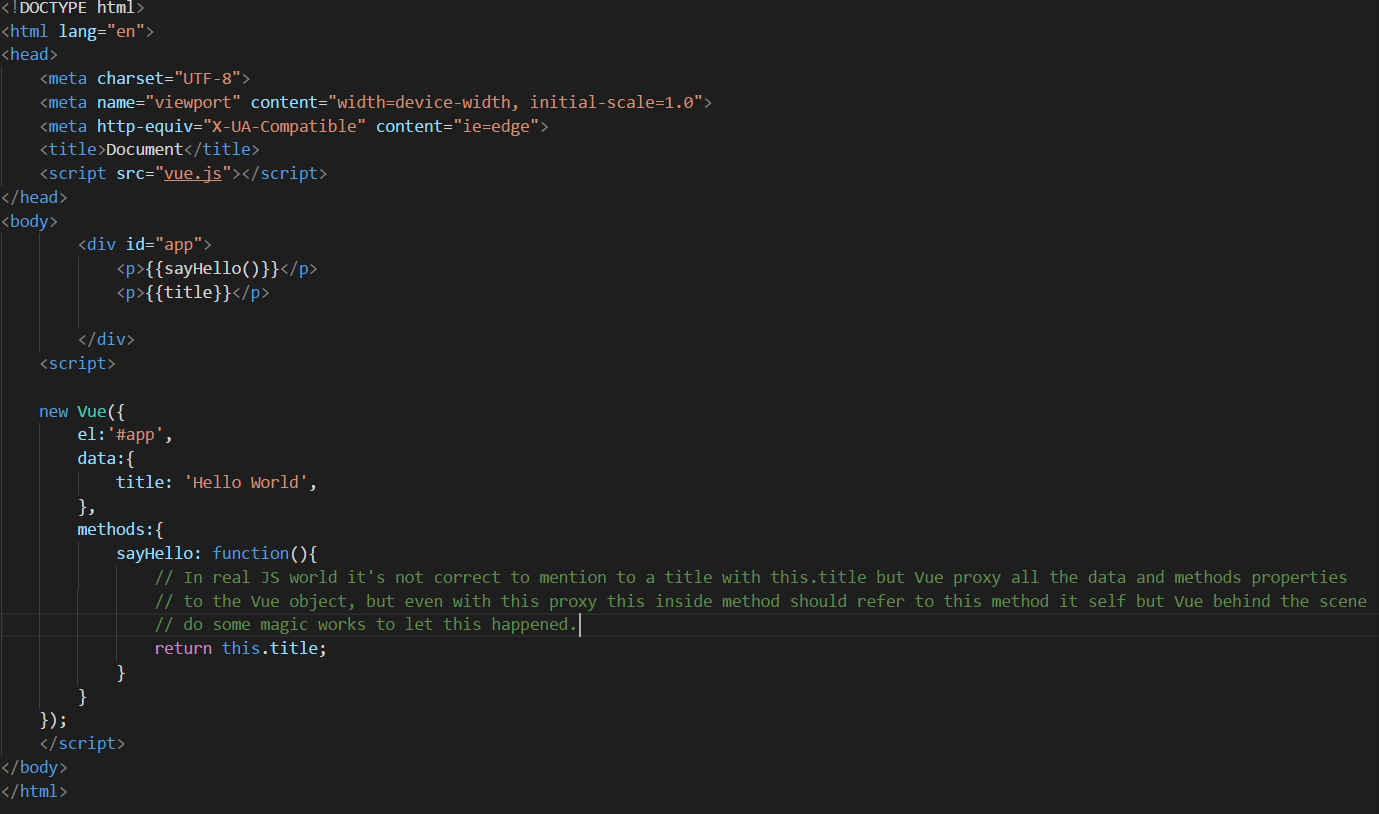


In addition to data properties, Vue instances expose a number of useful instance properties and methods. These are prefixed with $ to differentiate them from user-defined properties. For example:



In the future, you can consult the [**API reference**](https://vuejs.org/v2/api/#Instance-Properties) for a full list of instance properties and methods.

Important Note:



**DOM Interaction:**

# VueJs Template Syntax:

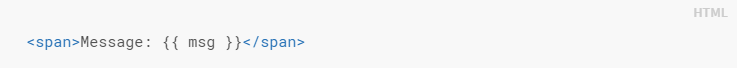
Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying Vue instance’s data. All Vue.js templates are valid HTML that can be parsed by spec-compliant browsers and HTML parsers.

Under the hood, Vue compiles the templates into Virtual DOM render functions. Combined with the reactivity system, Vue is able to intelligently figure out the minimal amount of components to re-render and apply the minimal amount of DOM manipulations when the app state changes.

If you are familiar with Virtual DOM concepts and prefer the raw power of JavaScript, you can also [**directly write render functions**](https://vuejs.org/v2/guide/render-function.html) instead of templates, with optional JSX support.

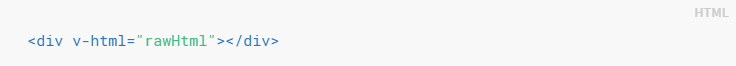
Example:

The most basic form of data binding is text interpolation using the “Mustache” syntax (double curly braces):



The mustache tag will be replaced with the value of the msg property on the corresponding data object. It will also be updated whenever the data object’s msg property changes.

You can also perform one-time interpolations that do not update on data change by using the [**v-once directive**](https://vuejs.org/v2/api/#v-once), but keep in mind this will also affect any binding on the same node:



# Accessing functions using {{}}:

# 

So, it’s also OK to use functions from methods object like data object within templating syntax, but you should be sure that toString() of this functions behaves to output what you need.

# v-bind:attr directive to bind on attributes :

# 

# Directives:

Directives are special attributes with the v- prefix. Directive attribute values are expected to be **a single JavaScript expression** (with the exception for v-for, which will be discussed later). A directive’s job is to reactively apply side effects to the DOM when the value of its expression changes. Let’s review the example we saw in the introduction:



Here, the v-if directive would remove/insert the <p> element based on the truthiness of the value of the expression seen.

* Arguments:

Some directives can take an “argument”, denoted by a colon after the directive name. For example, the v-bind directive is used to reactively update an HTML attribute:



Here href is the argument, which tells the v-bind directive to bind the element’s href attribute to the value of the expression url.

Another example is the v-on directive, which listens to DOM events:



Here the argument is the event name to listen to. We will talk about event handling in more detail too.

* Modifiers:

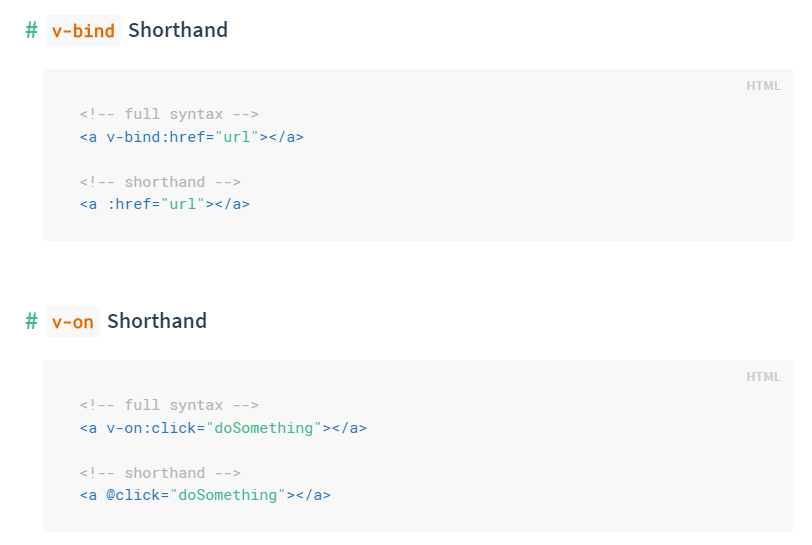
Modifiers are special postfixes denoted by a dot, which indicate that a directive should be bound in some special way. For example, the .prevent modifier tells the v-on directive to call event.preventDefault() on the triggered event:



We will see more use of modifiers later when we take a more thorough look at v-on and v-model.

* Shorthand:

The v- prefix serves as a visual cue for identifying Vue-specific attributes in your templates. This is useful when you are using Vue.js to apply dynamic behavior to some existing markup, but can feel verbose for some frequently used directives. At the same time, the need for the v- prefix becomes less important when you are building an [**SPA**](https://en.wikipedia.org/wiki/Single-page_application) where Vue.js manages every template. Therefore, Vue.js provides special shorthands for two of the most often used directives, v-bind and v-on:

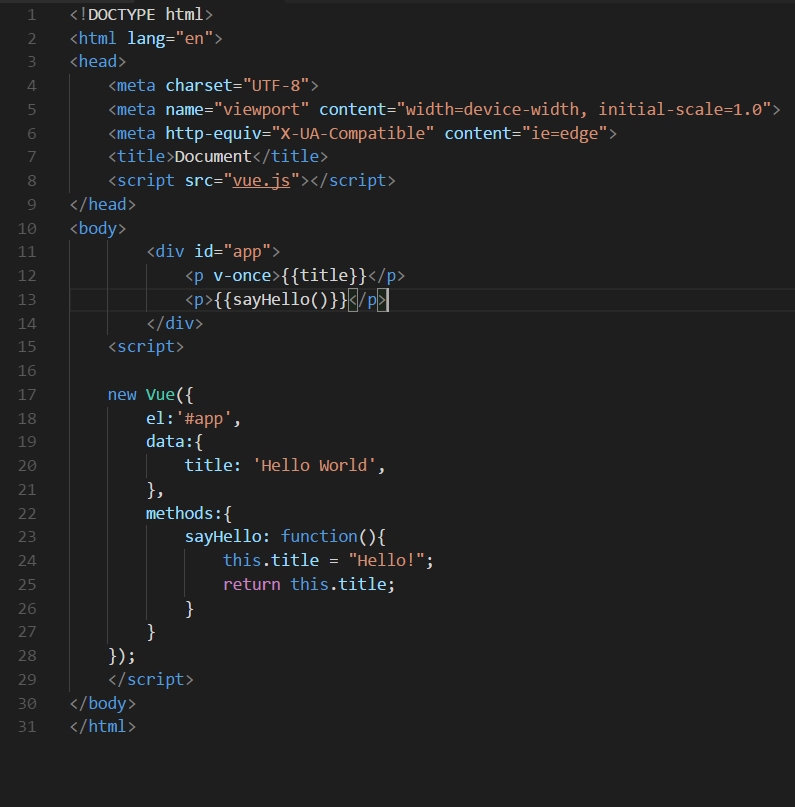
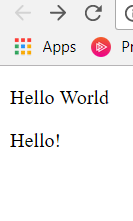


They may look a bit different from normal HTML, but : and @ are valid chars for attribute names and all Vue.js supported browsers can parse it correctly. In addition, they do not appear in the final rendered markup. The shorthand syntax is totally optional, but you will likely appreciate it when you learn more about its usage later.

# Disable re-render with v-once:

v-once make it very easy to stick to the initial value.

As shown below, if we don’t use v-once the title should be render as Hello! Because when sayHello() called at line 13 it changed the title value to Hello but by using v-once we stacked to the initial value of title.

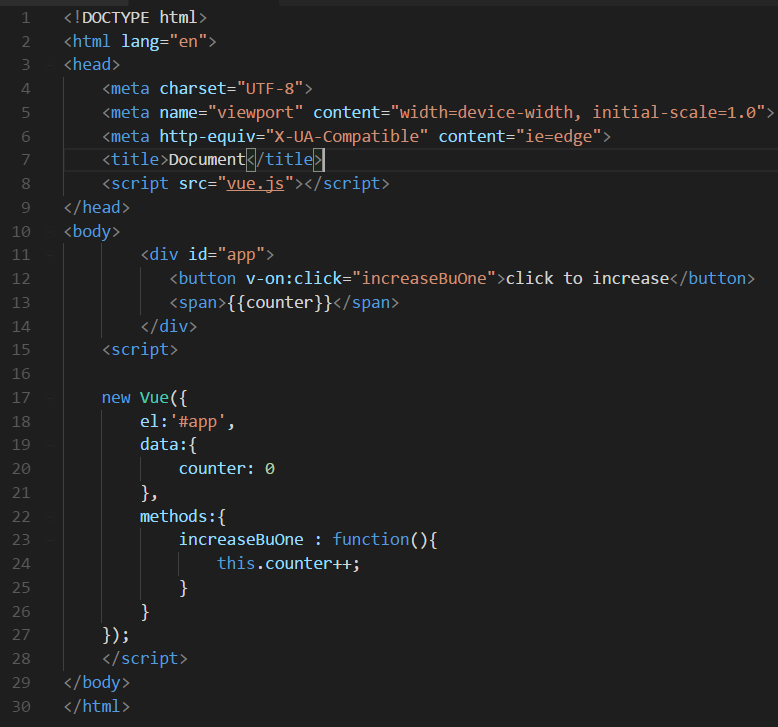


# Use v-html to output a raw html:



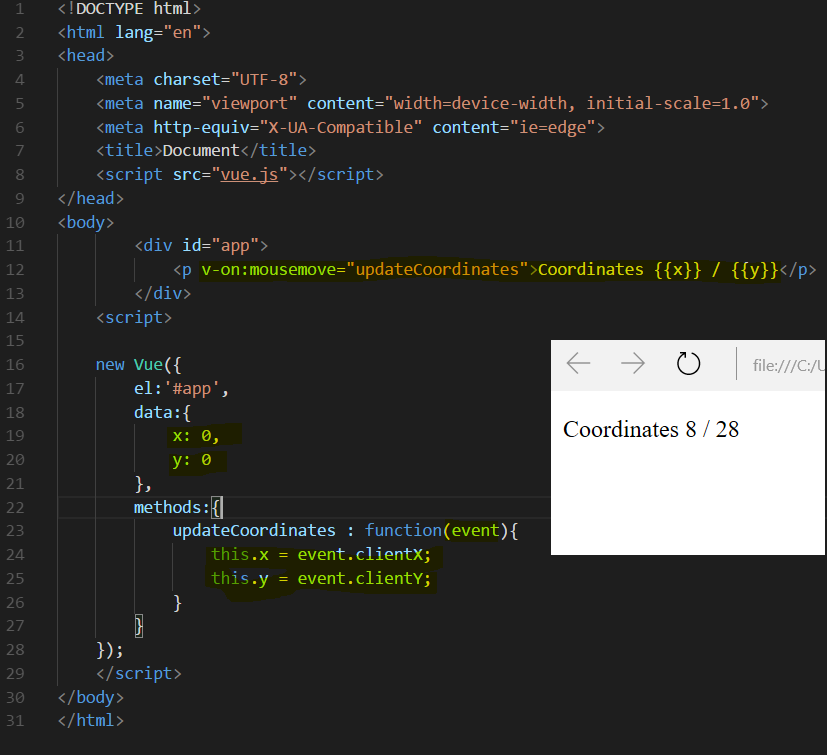
# Events:

# Listening to events:



# Getting event data from Event object:

As shown here below we can pass the event object of the event itself and we can use its properties easily.



# Passing your own arguments with events:

As shown here below we can pass our own parameters in addition to the event object of the event itself but here we should explicitly specify the event argument by the keyword $event as I passed with it addition parameters.

# 

# Events Modifiers:

Modifiers are special postfixes denoted by a dot, which indicate that a directive should be bound in some special way. For example, the .prevent modifier tells the v-on directive to call event.preventDefault() on the triggered event:

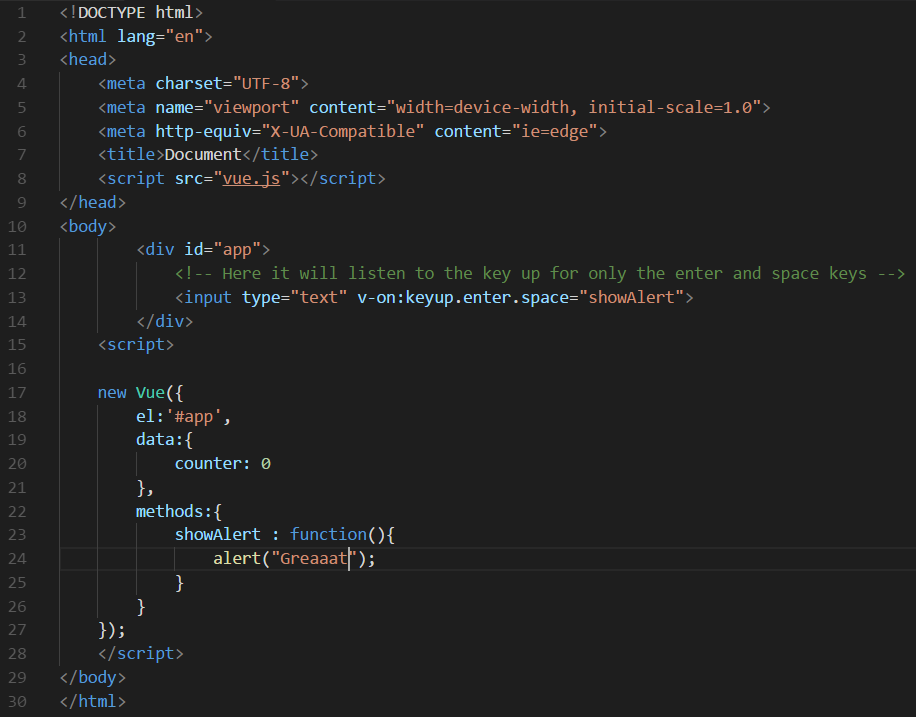


We will see more use of modifiers later when we take a more thorough look at v-on and v-model.

* Available modifiers: <https://vuejs.org/v2/guide/events.html#Event-Modifiers>

# Key Modifiers:

We can also add key modifiers like the below example:



Available key modifiers: <https://vuejs.org/v2/guide/events.html#Key-Modifiers>

# Remember:

you can always access the default JS Event object, which is created automatically, via $event in the code triggered by your event (=on the right side of the equal sign).

