Vue.js

Declarative Component-Based UI Framework

History of Vue.js

- Created by Evan You (ex-Googler)
- Version 0.6: Dec 2013 (first version on GitHub)
- Oct 2015: version 1.0
- 2016-2018: vers 2.0-2.5
- 2019-2021: vers 2.6.0-2.6.14
- 2020-2022: version 3.0-3.2
- Latest version: 3.2.31

Related Background:

Custom (HTML) Elements/ W3C Web Components

Why Use Web Components?

Native HTML

```
<body>
 <!-- menu --->
 <ul>
   <li><a href="#home">Home</a></li>
   <li><a href="#promo">Weekly Deals</a></li>
   <li><a href="#search">Search</a></li>
   <li><a href="#orders">Orders</a></li>
   <li><a href="#login">Signin</a></li>
 </ul>
 <div id="homescreen">
   <!-- details of home screen here -->
   <table>
     <tr>_____</tr>
     <tr>_____</tr>
   </table>
 </div>
 <div id="promoscreen">
   <!-- details of home screen here →
   <span>Don't miss this one-time offer:</span>
   <ol>
     <li>
   </ol>
 </div>
 <div id="searchscreen">
   <span>What are you looking for?</span>
   <form _____>
   </form>
 </div>
</body>
```

With Web Components

```
<body>
  <main-menu>
    <menu-item>Home</menu-item>
    <menu-item>Promotion</menu-item>
    <menu-item>Search</menu-item>
    <menu-item>Orders</menu-item>
    <menu-item>Signin</menu-item>
  </main-menu>
  <page-tabs>
    <tab-item><home-screen></tab-item>
    <tab-item><promo-screen></tab-item>
    <tab-item><search-prod></tab-item>
    <tab-item><order-list></tab-item>
    <tab-item><sign-in></tab-item>
  <page-tabs>
</body>
```

Which Front End Framework?

State of JS surveys:
2018 2019 2020 2021

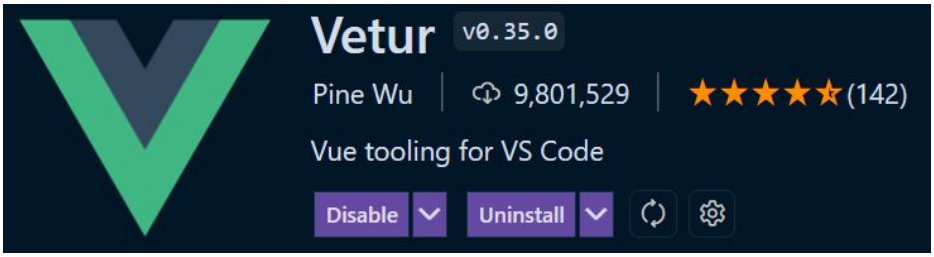
Resources

http://vuejs.org/guide (v1, v2, v3)
https://vuemastery.com

Which Vue version?

- We will cover mostly Vue 2.6.x
  - Many options for UI third-party libraries
  - These UI libraries are being ported to support Vue 3.x
- New features in Vue 3.x
  - Separate configuration settings per instance: easier to create multiple instances of VueJS within one web app
  - Composition API
    - Easier to maintain large components
    - Easier to maintain shared logic across multiple components
  - Reactive references

Recommended Extensions



- Syntax highlighting
- Linting/ Error Checking
- Formatting
  - Work seamlessly with Prettier
  - Common config file (for multi-developer teams) in `.vscode/settings.json`
- IntelliSense

Using vue-cli

- Install Node.js
- Install yarn
- Run one of the following commands to install vue-cli (**once per computer**)

```
npm i -g @vue/cli
# OR
yarn global add @vue/cli

vue --version      # @vue/cli 5.0.x or newer
```


npm vs. yarn

NPM	Yarn
`npm init -y`	`yarn init -y`
`npm install`	`yarn install`
`npm install` *`package-name`*	`yarn add` *`package-name`*
`npm install -save` *`package-name`*	`yarn add` *`package-name`*
`npm install -save-dev` *`package-name`*	`yarn add -D` *`package-name`*
`npm run` *`name-of-script`*	`yarn` *`name-of-script`*

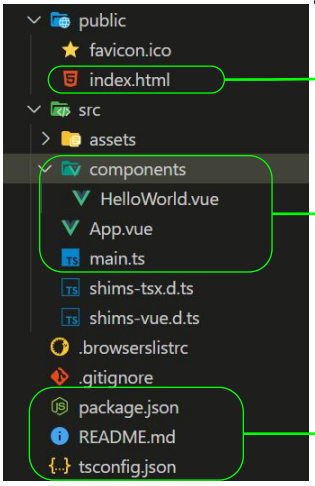
Create and run a new Vue App

```
vue create my-project
  ? Please pick a preset
    > Manually select features
  ? Check the features needed . . .
    (*) Babel
    (*) Typescript
    (*) Linter/Formatter
  ? Choose a version of Vue.js . . .
    (*) 2.x
  ? Use class-style component syntax? Y
  ? Use Babel alongside TS ___? Y
  ? Pick a linter / formatter config:
    > ESLint with error prevention only
  ? Pick additional lint feature:
    > Lint on save
  ? Where do you prefer placing config ___?
    > In package.json
  ? Save this as a preset ____? N
```

```
cd my-project
npm run serve
# or
yarn serve
```

Browser http://localhost:8080

What's included by vue-cli?



Webapp landing page

Main entry point and Vue components

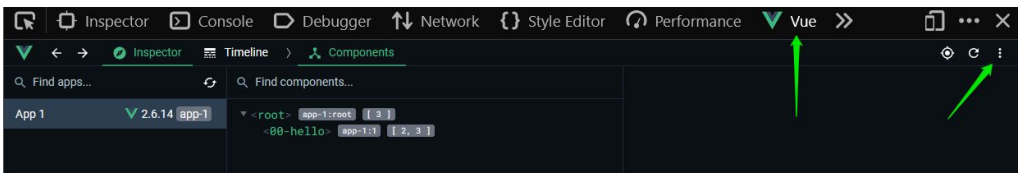
- *One index.html*
- *One main.ts*
- *Many *.vue files*
- *Optionally many *.ts files*

Project settings

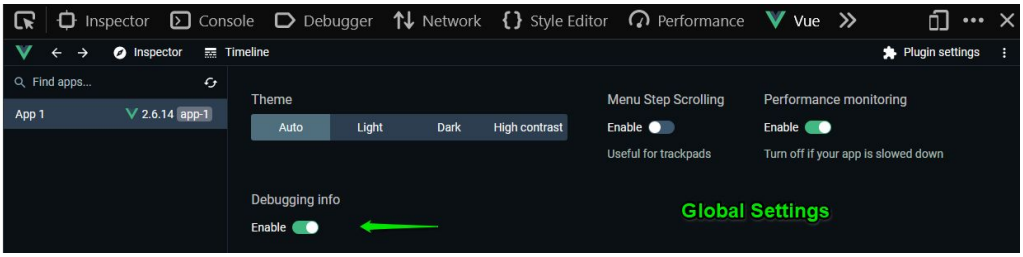
Install Vue DevTools (browser extension)

Vue DevTools

Hello VueJS



Hello VueJS



Live Demo
(a) Vue DevTools AND (b) Browser Debugger

Hello World (Simplified)

public/index.html

```
<html>
  <body>
    Begin
    <div id="vue-app"></div>
    End
  </body>
</html>
```

src/main.ts

```
import Vue from "vue";
import App from "./App.vue";

const app = new Vue({
  render:(h) => h(Hello);
  }
});
app.$mount("#vue-app");
```

src//App.vue

```
<template>
  <h1>Hello VueJS</h1>
<template>
```

Begin
Hello VueJS
End

Typical project organization:

- ***One index.html***
- ***One main.ts***
- ***Many xxxx.vue files***

Anatomy of a .vue (SFC = Single File Component)

src/SampleComponent.vue

```
<template>
  <div>
     <!-- The UI design in HTML goes here -->
  </div>
</template>

<script lang="ts">
import Vue from "vue"
import {Component} from "vue-property-decorator";

@Component
export default class SampleComponent extends Vue {
  // Your data and methods/functions
}
</script>

<style scoped>
// CSS style rules applied to the HTML template above
</style>
```

.vue = .html + (.ts | .js) + (.css | .scss | .sass)

The HTML elements inside the <template> must have a SINGLE root, typically a but can use any other grouping element

The class name has no significant meaning. You may name the class anything, but the common practice is to use the same name as the filename (without .vue)

*"**scope**" attribute implies the style will be applied only to this component and NOT to the child components of this one*

VSCode Demo #1
Initial Setup & 00-hello.vue

Code Snippet Examples

src//SampleComponent.vue

```
<template>
  <div>
     <!-- The UI design in HTML goes here -->
  </div>
</template>

<script lang="ts">
import Vue from "vue"
import {Component} from "vue-property-decorator";

@Component
export default class SampleComponent extends Vue {
  // Your data and methods/functions
}
</script>

<style scoped>
// CSS style rules applied to the HTML template above
</style>
```

To save screen space, the imports and @Component annotation will NOT be included in most of code snippet examples

1-way Data Binding
From class variables to UI

Data Binding: TS Expressions to UI

src//Hello.vue

```
<template>
  <h1>Hello {{who}} {{31 + 17}}</h1>
<template>

<script lang="ts">
export default class Hello extends Vue {
  private readonly who = "VueJS";
}
</script>
```

Begin
Hello VueJS 48
End

src//Hello.vue

```
<template>
  <h1>Hello {{who.toUpperCase()}}</h1>
<template>

<script lang="ts">
export default class Hello extends Vue {
  private readonly who = "VueJS";
}
</script>
```

Begin
Hello VUEJS
End

VueJS 2.x: Single Root Restriction

src//Hello.vue

```
<template>
  <h2>Introduction</h2>
  <p>Just say hello</h2>
</template>

<script lang="ts">
// Code not shown
</script>
```

Can't have TWO elements (<h2> and <p>) at the root level

src//Hello.vue

```
<template>
  <div>
    <h2>Introduction</h2>
    <p>Just say hello</h2>
  </div>
</template>

<script lang="ts">
// Code not shown
</script>
```

Enclose them in a single wrapper element (does not have to use)

Vue Data Binding Directives

HTML Attributes for Data Binding Directives

- `v-bind`: bind Vue data to HTML (native) attribute
- `v-for`: repeat data from arrays/lists
- `v-if`, `v-else`, `v-else-if`, `v-show`: conditional rendering
- `v-model`: 2-way data binding (data ⇌ UI)
  - Compare it to 1-way binding `{{my_data}}`
- And many more: `v-text`, `v-html`, `...`

v-bind: bind (Vue) data to HTML attributes

src//Hello.vue

```
<template>
  <div>
   <img src="http://bit.ly/10923f8d998.png">
   <img v=bind:src="imgLocation">
   <img :src="imgLocation">
  </div>
<template>
<script lang="ts">
export default class Hello extends Vue {
  private readonly imgLocation = "https://bit.ly/10923f8d998.png";
}
</script>
```

```
// Compare to the following code snippet
const imgLocation = https://bit.ly/10923f8d998.png"
const imgEl = document.createElement("img");
imgEl.setAttribute(src, imgLocation);
```


{{*data*}} binds data/var to text nodes

v-bind:*attr*="*data*" binds data/var to HTML attrs

VSCode Demo #2
10-data-binding & Vue DevTools

Iterate over arrays: v-for

src//Chemical.vue

```
<template>
  <div>
    <h1>Chemical Elements</h1>
    <ol>
      <li v-for="(a,arrIdx) in atoms"
          v-bind:key="arrIdx">{{a}}</li>
    </ol>
  </div>
<template>
<script lang="ts">
export default class Chemical extends Vue {
  private readonly atoms = ["Argon", "Barium", "Carbon"]
}
</script>
```

Use one of the syntax options

```
<li v-for="(a,pos) in atoms" v-bind:key="pos">{{a}}</li>
<li v-for="(a,pos) in atoms"       :key="pos">{{a}}</li>
```

```
<ol>
  <li>Argon</li>
  <li>Barium</li>
  <li>Carbon</li>
</ol>
```

Rendered Output

Chemical Elements

1. Argon
2. Barium
3. Carbon

- *V-for must be used together with :key*
- *:key is required for improved VueJS rendering performance*
- *:key must be a unique value (of **any data type**) **among siblings***

More on v-for :key

```
export default class XYZ extends Vue {
  private atoms = [
      {symbol: “Ar”, name: “Argon”},
      {symbol: “C”, name “Carbon”},
      {symbol: “Ne”, name: “Neon”}
    ]
}
```

*Keys must be **unique among siblings** (like primary keys in DB)*

```
<ul>
  <li v-for=”a in atoms”
      :key=”a.symbol”>{{a.name}}</li>
</ul>
```

```
<ul>
  <li v-for=”(a,pos) in atoms”
      :key=”pos”>{{a.name}}</li>
</ul>
```


Repeating a group of elements (*the wrong way*)

src//Movie.vue

```
<template>
  <div>
    <h2 v-for=”(m,pos) in movies”
        :key=”pos”>{{m.title}}</h2>
    <p v-for=”(m,pos) in movies”
        :key=”pos”>Release year:{{m.year}}</p>
  </div>
<template>
<script lang=”ts”>
@Component
export default class Movie extends Vue {
  private readonly movies = [
  {title: “Batman Begins”, year: 2005},
  {title: “The Upside”, year: 2017}]
}
</script>
```

HTML output

```
<div>
  <h2>Batman Begins</h2>
  <h2>The Upside</h2>
  <p>Release year: 2005</p>
  <p>Release year: 2017</p>
</div>
```

Rendered Output

Batman Begins
The Upside
Release year: 2005
Release year: 2017

Repeating a group of elements (*almost correct*)

src//Hello.vue

```
<template>
  <div>
    <template v-for="(m,pos) in movies">
      <h2 :key="pos">{{m.title}}</h2>
      <p  :key="pos">Release year:{{m.year}}</p>
    </template>
  </div>
<template>
<script lang="ts">
export default class Movie extends Vue {
  private readonly movies = [
  {title: "Batman Begins", year: 2005},
  {title: "The Upside", year: 2017}]
}
</script>
```

internal VueJS DOM

```
<div>
  <h2 key="0">Batman Begins</h2>
  <p  key="0">Release year: 2005</p>
  <h2 key="1">The Upside</h2>
  <p  key="1">Release year: 2017</p>
</div>
```

```
[Vue warn]: Duplicate keys detected: '0'. This may cause an update error.  vue.runtime.esm.js:619

found in

---> <Movie>
       <Root>

[Vue warn]: Duplicate keys detected: '1'. This may cause an update error.  vue.runtime.esm.js:619

found in

---> <Movie>
       <Root>
```


Repeating a group of elements

src//Hello.vue

```
<template>
  <div>
    <template v-for="(m,pos) in movies">
      <h2 :key="`a-${pos}`">{{m.title}}</h2>
      <p :key="`b-${pos}`">Release year: {{m.year}}</p>
    </template>
  </div>
<template>
<script lang="ts">
export default class Movie extends Vue {
  private readonly movies = [
    {title: "Batman Begins", year: 2005},
    {title: "The Upside", year: 2017}]
}
</script>
```

internal VueJS DOM

```
<div>
  <h2 key="a-0">Batman Begins</h2>
  <p key="b-0">Release year: 2005</p>
  <h2 key="a-1">The Upside</h2>
  <p key="b-1">Release year: 2017</p>
</div>
```

- *Use v-for on the <template>*
- *But apply the :key to the **actual** elements*
- *Use backquotes for string interpolation*

Rendered Output

Batman Begins
Release year: 2005

The Upside
Release year: 2017

:key details

```
<template v-for="(_, pos) in ____">

  <h2 :key=" `a-${pos}` ">_____</h2>
  <p  :key=" `b-${pos}` ">_____</p>

</template>
```

Lots of quotes:
- *Backquotes for ${variable} interpolation*
- *Double quotes to enclose the entire string expression*

Repeat N times: v-for

src//Hello.vue

```
<template>
  <div>
    <h1>Count Down</h1>
    <p v-for="n in 5" v-bind:key="n">Downto {{6-n}}...</p>
  </div>
<template>
```

- *The loop variable (n in the above snippet) starts at ONE (not zero)*
- *Variables can be used in place of the constant (number 5)*

Rendered Output

Count Down

Downto 5...
Downto 4...
Downto 3...
Downto 2...
Downto 1...

VSCode Demo #3
20-forloop & 22-forloop-object

Updating Arrays

Detected by VueJS

```
// Replace the ENTIRE array
this.myArray = [ /* provide a new array */ ];

// Add new item
this.myArray.push(provide_a_new_item);

// Remove the last item
this.myArray.pop();

// Remove (one item) at the k-th position
this.my.array.splice(k, 1);
```

NOT detected by VueJS

```
// Replace a specific item (Wrong way)
this.myArray[k] = new_content;
```

Detected by VueJS

```
// Replace a specific item (acceptable)
this.myArray.splice(k, 1, new_content);
```

"Detected by VueJS" means the UI that depends on the array will be refreshed correctly

Conditional: v-if, v-else-if, v-else

src//Random.vue

```
<template>
  <div>
    <h2>Random number: {{randVal}}</h2>
    <p v-if="randVal < 31">Below 31</p>
    <p v-else-if="randVal > 87">87 or more</p>
    <p v-else>Between 32-87</p>
  </div>
<template>
<script lang="ts">
export default class Random extends Vue {
  private randVal = 1 + Math.floor(
    Math.random() * 100);    // 1-100
</script>
```

Rendered Output

Random number: 49

Between 32-87

```
<div>
  <h2>Random number: 49</h2>
  <p>Between 32-87</p>
</div>
```

Rendered Output

Random number: 14

Below 31

```
<div>
  <h2>Random number: 14</h2>
  <p>Below 31</p>
</div>
```

These directives automatically suppress elements whose condition evaluates to FALSE

Group Conditional

poor

```
<template>
  <div>
    <h2>Introduction</h2>
    <p>Lorem ipsum ...blah...blah...</p>
    <h2 v-if="!loggedIn">Logging in to your Account</h2>
    <p v-if="!loggedIn">Please do the following___</p>

    <p>Thank you!</p>

  </div>
</template>
```

better

```
<template>
  <div>
    <h2>Introduction</h2>
    <p>Lorem ipsum ...blah...blah...</p>
    <template v-if="!loggedIn">
      <h2>Logging in to your Account</h2>
      <p>Please do the following___</p>
    </template>
    <p>Thank you!</p>
  </div>
</template>
```

v-if vs. v-show

- *v-if does NOT generate the element(s) when the condition is false*
- *v-show generates the element(s) and use CSS to hide them*

src//Sample.vue

```
<template>
  <div>
    <p>Welcome</p>
    <p v-if="debugMode">Use STOP to kill the app</p>
    <p>Good bye!</p>
  </div>
</template>
<script lang="ts">
export default class Sample extends Vue {
  readonly debugMode = false;
</script>
```

rendered HTML

```
<div>
  <p>Welcome</p>
  <p>Good bye!<p>
</div>
```

src//Sample.vue

```
<template>
  <div>
    <p>Welcome</p>
    <p v-show="debugMode">Use STOP to kill the app</p>
    <p>Good bye!</p>
  </div>
</template>
<script lang="ts">
export default class Sample extends Vue {
  readonly debugMode = false;
</script>
```

rendered HTML

```
<div>
  <p>Welcome</p>
  <p style="visibility:none">Use STOP to kill the app</p>
  <p>Good bye!<p>
</div>
```


v-show and v-if on Custom Vue Elements

```
<template>
   <DeliveryMessage order-id="FY4328ZX" v-if="itemDelivered" />
</template>
<script lang="ts">
export default class Shopping extends Vue {
  private itemDelivered = false;
}
</script>
```

Since DeliveryMessage is NOT rendered, any initialization logic in DeliveryMessage.vue did NOT run

```
<template>
   <DeliveryMessage order-id="FY4328ZX" v-show="itemDelivered" />
</template>
<script lang="ts">
export default class Shopping extends Vue {
  private itemDelivered = false;
}
</script>
```

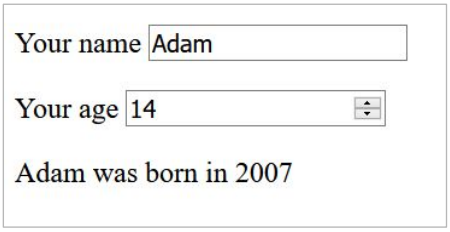
*Since DeliveryMessage is **rendered (but hidden)**, any initialization logic in DeliveryMessage.vue **would have RUN***

VS Demo #4: 30-conditional

Two-Way Data Binding (v-model)

src//Sample.vue

```
<template>
  <div>
    <p>Your name <input type="text" v-model="name"></p>
    <!--
    <p>Your name <input type="text" v-model.lazy="name"></p>
    -->
    <p>Your age <input type="number" v-model.number="age"></p>

    <p>{{name}} was born in {{thisYear - age}}</p>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  readonly thisYear: number = newDate().getFullYear();
  private name:string = "Adam";
  private age:number = 1;
</script>
```

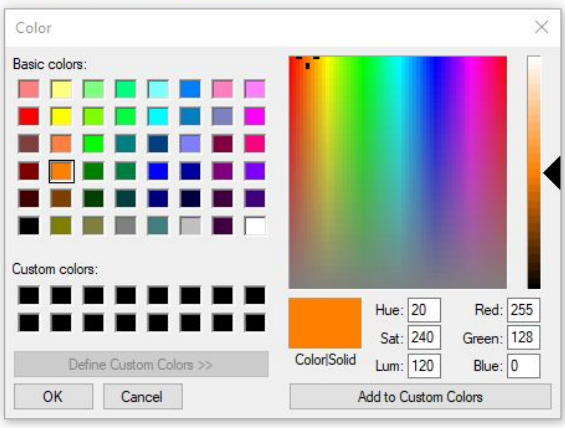


- *Input type email, password, color, date is handled similarly to type="text"*
- *Input type="range" (a horizontal slider) is handled similarly to type="number"*
- *Lazy: bind the value after input lost keyboard focus*

VSCode Demo #5
40-input-binding-simple

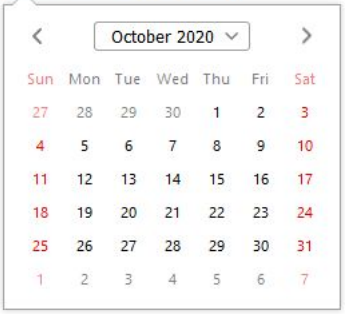

```
<input type="color" v-model="hexColorStr">
```

Pick a color:



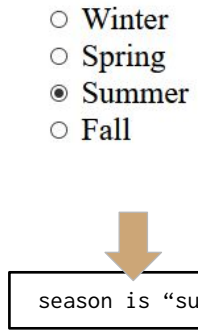

```
<input type="date" v-model="dateStr">
```

Pick a date mm / dd / yyyy



Radio buttons with string result

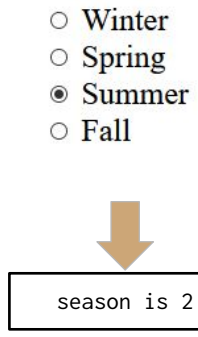
```
<template>
  <div>
    <input type="radio" id="t0" value="wi" v-model="season">
    <label for="t0">Winter</label>
    <input type="radio" id="t1" value="sp" v-model="season">
    <label for="t1">Spring</label>
    <input type="radio" id="t2" value="su" v-model="season">
    <label for="t2">Summer</label>
    <input type="radio" id="t3" value="fa" v-model="season">
    <label for="t3">Fall</label>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  private season:string;
</script>
```



Dropdown menus are handled similar to a radio button

Radio buttons with numeric result

```
<template>
  <div>
    <input type="radio" id="t0" value="0" v-model.number="season">
    <label for="t0">Winter</label>
    <input type="radio" id="t1" value="1" v-model.number="season">
    <label for="t1">Spring</label>
    <input type="radio" id="t2" value="2" v-model.number="season">
    <label for="t2">Summer</label>
    <input type="radio" id="t3" value="3" v-model.number="season">
    <label for="t3">Fall</label>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  private season:number;
</script>
```



Radio buttons with numeric result & data array

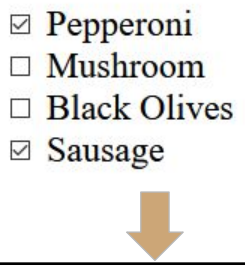
```
<template>
  <div>
    <input type="radio" value="0"
           v-model.number="season">
    <labe>Winter</label>
    <input type="radio" value="1"
           v-model.number="season">
    <label>Spring</label>
    <input type="radio" value="2"
           v-model.number="season">
    <label>Summer</label>
    <input type="radio" value="3"
           v-model.number="season">
    <label>Fall</label>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  private season:number;
</script>
```

v-for and array source

```
<template>
  <div>
    <template v-for="(s,pos) in allSeasons">
      <input type="radio" :value="pos" :key="`inp-${pos}`"
             v-model.number="season">
      <label :key="`lab-${pos}`">{{s}}</label>
    </template>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  allSeasons = ["Winter", "Spring", "Summer", "Fall']
  private season:number;
</script>
```

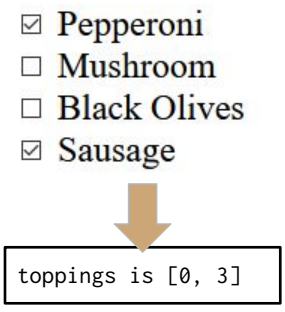

Checkbox with string result

```
<template>
  <div>
    <input type="checkbox" id="t0" value="pep" v-model="toppings">
    <label for="t0">Pepperoni</label>
    <input type="checkbox" id="t1" value="msh" v-model="toppings">
    <label for="t1">Mushroom</label>
    <input type="checkbox" id="t2" value="blo" v-model="toppings">
    <label for="t2">Black Olives</label>
    <input type="checkbox" id="t3" value="sau" v-model="toppings">
    <label for="t3">Sausage</label>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  private toppings:string[];
</script>
```



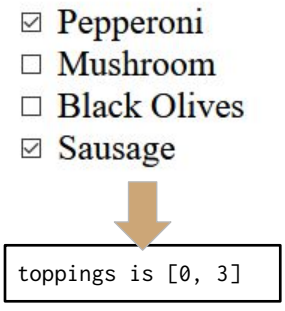
Checkbox with numerical result

```
<template>
  <div>
    <input type="checkbox" id="t0" value="0" v-model.number="toppings">
    <label for="t0">Pepperoni</label>
    <input type="checkbox" id="t1" value="1" v-model.number="toppings">
    <label for="t1">Mushroom</label>
    <input type="checkbox" id="t2" value="2" v-model.number="toppings">
    <label for="t2">Black Olives</label>
    <input type="checkbox" id="t3" value="3" v-model.number="toppings">
    <label for="t3">Sausage</label>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  private toppings:number[];
</script>
```



Checkbox with numerical result (+ data array)

```
<template>
  <div>
    <input type="checkbox" id="t0" value="0" v-model.number="toppings">
    <label for="t0">Pepperoni</label>
    <input type="checkbox" id="t1" value="1" v-model.number="toppings">
    <label for="t1">Mushroom</label>
    <input type="checkbox" id="t2" value="2" v-model.number="toppings">
    <label for="t2">Black Olives</label>
    <input type="checkbox" id="t3" value="3" v-model.number="toppings">
    <label for="t3">Sausage</label>
  </div>
</template>

<script lang="ts">
@Component
export default class Sample extends Vue {
  private toppings:number[];
</script>
```



VSCode Demo #6
42-input-binding

Event Handling

VueJS ref vs. HTML id

```
<template>
  <div>
    <span id="foo" ref="bar">Some text</span>
  <div>
</template>

<script lang="ts">
export default class Sample extends Vue {
  doWork() {
    const spElem = this.$refs.bar as Element;
    // code that manipulates spElem goes here
  }
}
</script>

<style>
#foo {
  min-width: 20em;
}
</style>
```

- *Use ID to refer elements in <style>*
- *Use REF to refer to elements in <script>, typically inside event handling functions*
- *this.$refs is a special VueJS array/object that holds all the REF in the current file*

Event Handling Directives

```
<someHTMLTag v-on:domEvents="yourEventHandlingFunction"  ...>

<someHTMLTag @domEvents="yourEventHandlingFunction"  ...>
```

Event Handling Functions

```
<template>
  <div>
    <button v-on:click="handleGo">Go</button>
    <!-- OR -->
    <button @click="handleGo">Go</button>
    <p>Click count {{count}}</p>
  </div>
</template>

<script lang="ts">
export default class Sample extends Vue {
  private count: number = 0;

  handleGo(ev:MouseEvent) {
    this.count++;
  }
}
</script>
```

@click is a shortcut of **v-on:click**

Multiple event handlers on one element

```
<template>
  <div>
    <span @mouseenter="goIn" @mouseleave="goOut">
    </span>
  </div>
</template>

<script lang="ts">
export default class Sample extends Vue {
  goIn(ev:MouseEvent) {
    /* your code here */
  }
  goOut(ev:MouseEvent) {
    /* your code here */
  }
}
</script>
```

Tons of Event Names

Event Names	Function Argument Type
keypress, keydown, keyup, key	KeyboardEvent
wheel	WheelEvent
click	MouseEvent
blur, focus	FocusEvent
mousedown, mouseenter, mousemove, mouseup	MouseEvent

Mouse/Keyboard Events: Filters/Modifiers

```
<template>
  <div>
   <input type="text" @keydown.right="showNextPage"
       @keydown.alt.left="showFirstPage">
   <button @click.shift="goFirst">Start Over</button>
  </div>
<template>
<script lang="ts">
export default class Sample extends Vue {
  showNextPage(ev:KeyboardEvent) {
    /* called only when right arrow key is pressed */
  }
  showFirstPage(ev:KeyboardEvent) {
    /* called only when alt-left arrow key is pressed */
  }
  goFirst(ev:MouseEvent) { /* code here */ }
}
</script>
```

Filters:

.enter
.tab
.delete
.esc
.space
.up
.down
.left
.right

Modifiers:

.alt
.ctrl
.meta
.shift

VS Demo #7:
50-event-handling & 55-event-mods

Vue 2.x Lifecycle Functions

- beforeCreate() / created() ⇒ “constructor”
- beforeMount() / mounted()
  - The component is about to/has been inserted to the DOM tree
- beforeUpdate() / updated()
  - The component data is about to/has changed and UI will refresh
  - Do not update the component’s data in these functions. You’ll stuck in an infinite “loop”
- beforeDestroy() / destroyed()
  - The component is about to/has been removed from the DOM tree
- https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram

Lifecycle Functions

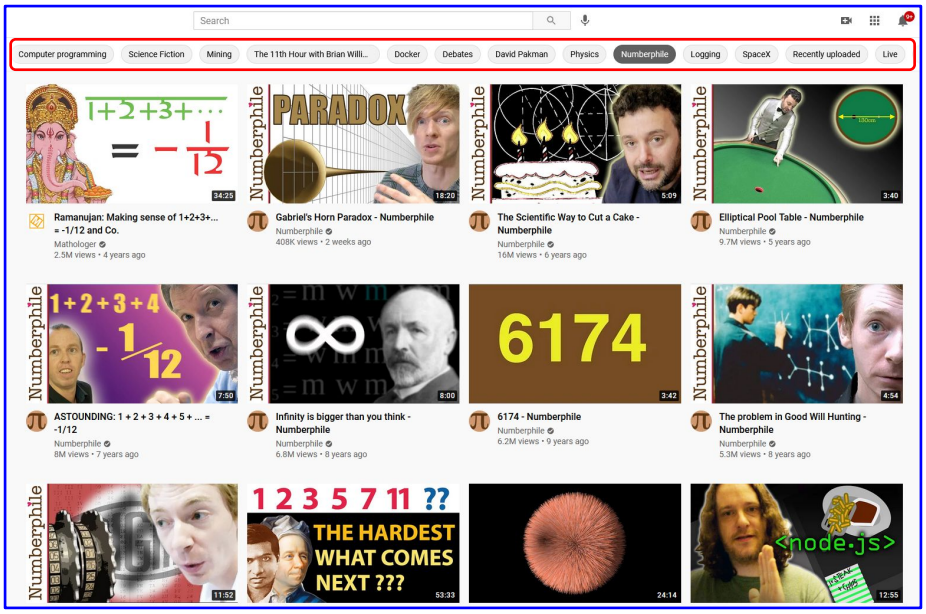
```
<template><!-- blah blah--></template>

<script lang="ts">
export default class Sample extends Vue {
  private yourData: ___;
  beforeCreate() { /* called before <Sample> is created */ }
  created() { /* called AFTER <Sample> is created */ }
  beforeMount() {
    // This will be called BEFORE <Sample> is inserted to the DOM tree
  }
  mounted() {
    // This will be called AFTER <Sample> is inserted to the DOM tree
  }
  beforeDestroy() {
    // This will be called BEFORE <Sample> is removed from the DOM tree
  }
  destroyed() {
    // This will be called AFTER <Sample> is removed from the DOM tree
  }
}
</script>
```

Use for initialization logic that that does NOT depend on DOM existence

Use for initialization logic that that depends on DOM existence

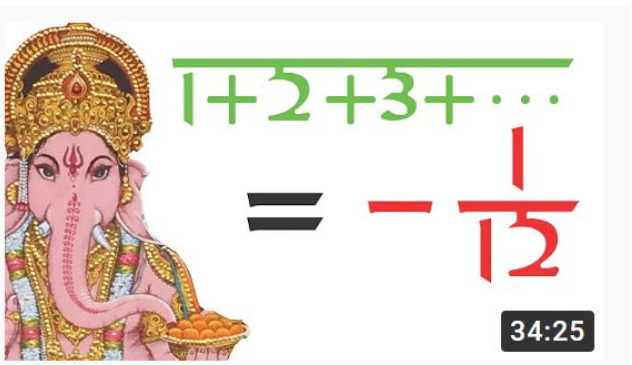
Use for "undoing" the effect of initialization above

Using Multiple Vue Components



Video Tag

Video Cover



coverImage

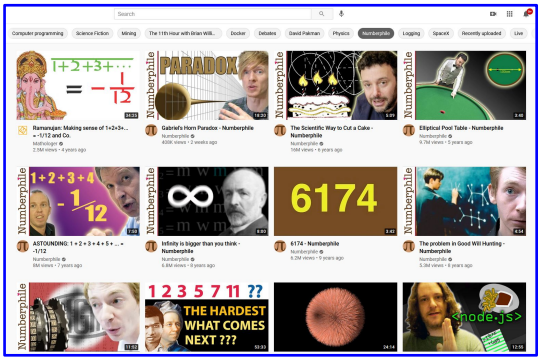
videoDuration

videoTitle

authorName

numberOfViews

releaseDate



src/YouTubeApp.vue

```
<template>
  <div>
    <YouTubeCover v-for="z in availableVideos"
       :cover-image="z.imageURL"
       :title="z.videoTitle"
       :duration="z.videoDuration"
       :views="z.numberOfViews"
     />
  </div>
</template>

<script lang="ts">
import YouTubeCover from "./components/YTCover.vue";

@Component({ components: {YouTubeCover} })
export default class YouTubeApp extends Vue {
  private availableVideos = [/* vide data here */];
}
</script>
```


src/YouTubeApp.vue

```
<template>
  <div>
    <YouTubeCover v-for="z in availableVideos"
       :cover-image="z.imageURL"
       :title      ="z.videoTitle"
       :duration   ="z.videoDuration"
       :views      ="z.numberOfViews"
     />
  </div>
</template>

<script lang="ts">
import YouTubeCover from "./components/YTCover.vue";

@Component({ components: {YouTubeCover} })
export default class YouTubeApp extends Vue {
  private availableVideos = [/* video data here */];
}
</script>
```

Parent Component

Child Component(s)

Kebab-case to camelCase conversion

src/components/YTCover.vue

```
<template>
  <div><!-- UI design goes here --></div>
</template>

<script lang="ts">
import {Component, Prop } from
   "vue-property-decorator";

@Component
export default class YTCover extends Vue {
  @Prop() readonly coverImage!:string;
  @Prop() readonly title!: string;
  @Prop() readonly duration!: number;
  @Prop() readonly views!: number;

}
</script>
```

kebab-case vs. camelCase

kebab-case (in HTML)	camelCase (in TypeScript)
image	image
cover-image	coverImage
cover-image-url	coverImageUrl

Passing Arguments to Event Handler

```
<template>
  <ul>
    <template v-for="p in planets">
      <li>___
      <button @click="deletePlanet">Delete</button>

      <button @click="showPlanet(p.name)">View</button>
      <button @click="showDetails($event)">Details</button>
      </li>
    </template>
  <ul>
</template>
```

```
<script lang="ts">
export default class Astronomy extends Vue {
  private readonly planets: [
    {name: "Mercury", revolution: 87.97},
    {name: "Earth", revolution: 365.26},
    {name: "Mars", revolution: 686.68}
  ]
  deletePlanet() {
    /* code here */
  }
  showDetails (ev:MouseEvent) {
    // code here
  }
  showPlanet (planetName:string) {
    // code here
  }
}
</script>
```

Vue.js Component (UI) Libraries

- Vuetify
- Quasar
- Element
- Vue Material
- Keen UI
- Buefy
- Bootstrap Vue
- Muse-UI
- AT-UI
- Vux
- iView
- Uiv
- Vuikit
- Onset UI + Vue
- Semantic Ui + Vue
- Fish-UI
- Mint UI
- Framework7 Vue
- Cube UI
- Vueblue
- Ant design Vue

Vue Router
(in a separate slide)

Vuex
(Separate Slide)