VUEJS CHEATSHEET FOR DEVELOPERS

Put together by your friends at learnvue.co

TEMPLATE SYNTAX

Text Interpolation Options

```
<span> {{ msg }} </span>
<span v-text='msg'></span>
```

Setting Inner HTML

```
<span v-html='rawHTML'></span>
```

Can use JS Expressions; NOT JS Statements

```
<span> {{ msg.reverse() }} </span>
<span> {{ let msg = 'hi' }} </span>
```

DIRECTIVES

`v-if`	Puts el in DOM if true
`v-else-if`	Like a usual conditional
`v-else`	Like a usual conditional
`v-show`	Toggles display CSS value
`v-text`	Sets the inner text
`v-html`	Sets the inner HTML
`v-for`	Loop through an array/obj
`V-on or @`	Listens to DOM events
`V-bind or :`	Reactive updates attribute
`v-model`	Two way data binding
`v-once`	Sets val once; Never update

CONDITIONAL RENDERING

Add/Remove Element from DOM w/ Boolean

```
<div v-if='date == today'>...</div>
<div v-else-if='!done'>...</div>
<div v-else>...</div>
```

Toggles display CSS instead of editing DOM

```
<div v-show='date == today'>...</div>
```

HANDLING EVENTS

Capture and event and call a method

```
<div v-on:click='count'>Increase</div>
<!-- SHORTHAND -->
<div @click='count'>Increase</div>
```

Method is passed a Native DOM Cvent

```
count: function (event) {
     console.log(event.target)
}
```

Event modifiers (usage: v-on:click.stop)

`.stop`	Stops event propagation
`.once`	Can only trigger event once
`.prevent`	Calls evt.preventDefault
`.self`	Don't send if target = child

LIST RENDERING

Basic Loop Over Array

```
<li v-for='item in items' :key='item'>
      {{ item }}
</li>
```

Loop and Track Index

```
<li v-for='(item, index) in items'>
      {{ index }} : {{ item }}
</li>
```

Loop Values in Object

```
<li v-for='obj in objects'>
      {{ obj }}
</li>
```

VUEJS CHEATSHEET FOR DEVELOPERS

Put together by your friends at learnvue.co

BINDING DATA

Simple Binding

```
<div v-bind:id='objectID'>...</div>
<!-- SHORTHAND -->
<div :id='objectID'>...</div>
```

Two way binding with data and input

```
<input v-model='email' />
```

Input Modifiers

`.lazy`	**updates on change event**
`.trim`	**removes extra whitespace**

Use Objects to Bind Class/Styles

```
<input :class='{error: hasError}' />
<input :style='{margin: space+"px"}' />
```

BIND DATA BETWEEN CHILD & PARENT

Use v-bind to pass data from parent to child and emit a custom event to send data back.

In Parent, Bind Data & Set Listener to Update

```
<custom :msg='s' @update='s = $event' >
```

In Child, Send Back Using $emit(event, data)

```
this.$emit('update', 'hello world')
```

SLOTS

Slots allow for content injection from a parent component to a child component.

BASIC SLOTS

Child Component (MyButton.Vue)

```
<div>
   Hello World
   <slot></slot>
</div>
```

Parent Component

```
<my-button>
   This content will replace the slot
</my-button>
```

NAMED SLOTS

Useful when you have multiple slots. If unnamed, name is 'default'.

Child Component (MyButton.Vue)

```
<div>
   <slot name='top'></slot>
   <slot name='bottom'></slot>
</div>
```

Name Slots in the Parent Component

```
<my-button>
   <template v-slot:top>...
   </template>
   <template v-slot:bottom>...
   </template>
</my-button>
```

SCOPED SLOTS

Give parent component access to child data.

Child Component (MyButton.Vue)

```
<div>
   <slot v-bind:post='post'>
      {{ post.title }}
   </slot>
</div>
```

Parent Has Access to MyButton post data

```
<my-button>
   <template v-slot:default='slotData'>
      {{ post.author }}
   </template>
</my-button>
```

VUEJS CHEATSHEET FOR DEVELOPERS

Put together by your friends at learnvue.co

VUEJS LIFECYCLE HOOKS

beforeCreate()	Start of component
create()	Reactive data exists
beforeMount()	Before mounting DOM
mounted()	DOM can be accessed
beforeUpdate()	Still have old values
updated()	Values have been changed
beforeDestroy()	Component still complete
destroyed()	Teardown complete

VUE LIFECYCLE METHODS

$mount()	Mount component to DOM
$forceUpdate()	Force re-render
$nextTick()	Runs func next update
$destroy()	Destroy component/app

VUE OBJECT OPTIONS

data()	Init reactive data
props	Data visible by parent
mixins	Declares mixins
components	Registers children
methods	Set of Vue methods
watchers	Watch values for change
computed	Cached reactive methods

COMPUTED METHOD

A computed function is a method that only updates when a value it's dependent on changes.

```
computed: {
 fullName: function () {
   return this.fName + ' ' + this.lNAme
 }
}
```

WATCHERS

Listens to a reactive value and triggers an event when it changes. It is useful when you need to trigger methods when data changes.

```
watch: {
    fName: function (newVal, oldVal) {
       this.msg = 'fName changed!'
    }
}
```

TOP VUE LIBRARIES

vue-cli	Command Line Interface
vue-router	Handles Routing for SPAs
vuex	State Management Library

GREAT VUE UI RESOURCES

Vuetify	Bootstrap Vue	UIV
VueStrap	Vue Material	Mint UI
Element UI	Vuecidity	iView
Buefy	DeepReader	KeenUI
Quasar	AT UI	Vulma
Fish-UI	Muse UI	Vue Blu

CONTACT

For any corrections, comments, or concerns, just contact me at matt@learnvue.co

Hope this helped!