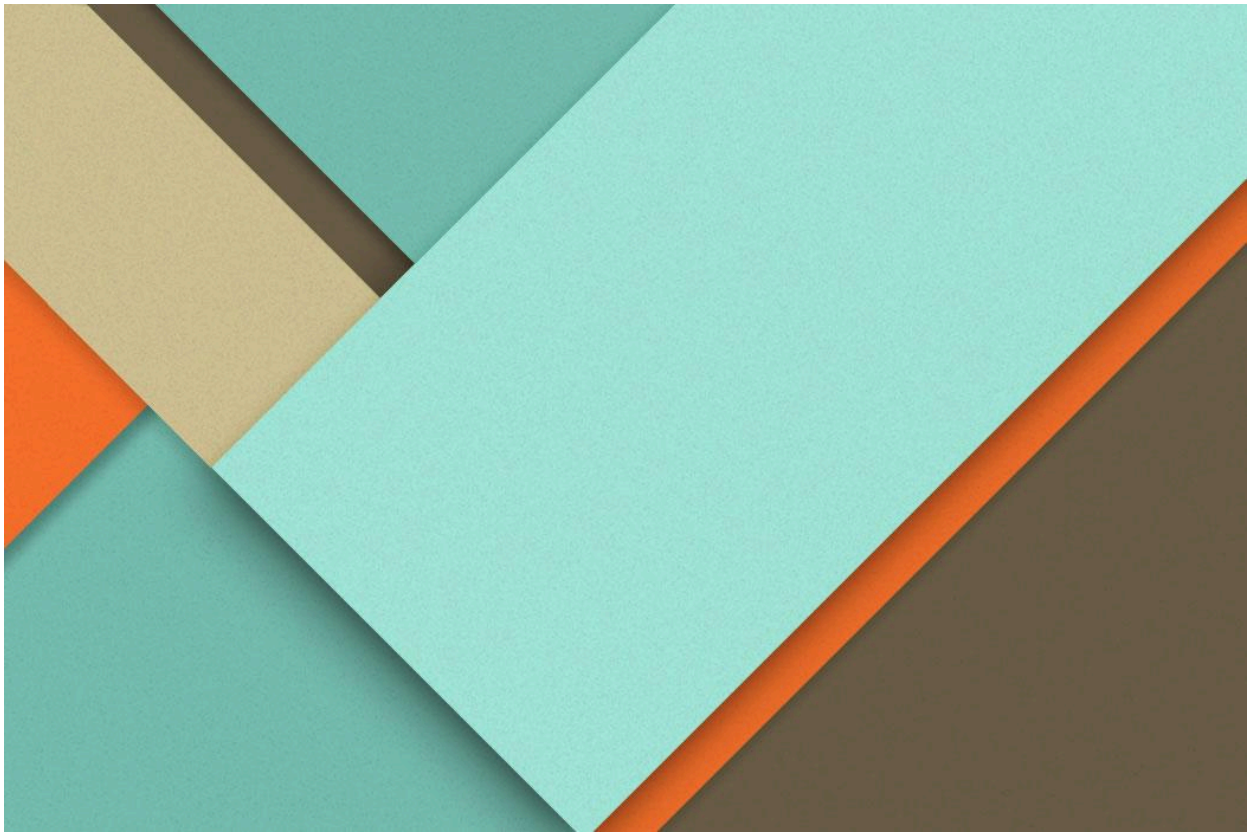


Web Development Cheatsheet

html, CSS, Vue, Nuxt, Javascript

—

Silvan
ArSiJa
Switzerland
silvan@arsija.net
arsija.net

About

A cheatsheet with all important information for Web Development, mainly for Vue/Nuxt.

Goals

1. Create an all in one Document for Web Development
2. Make it Printer friendly

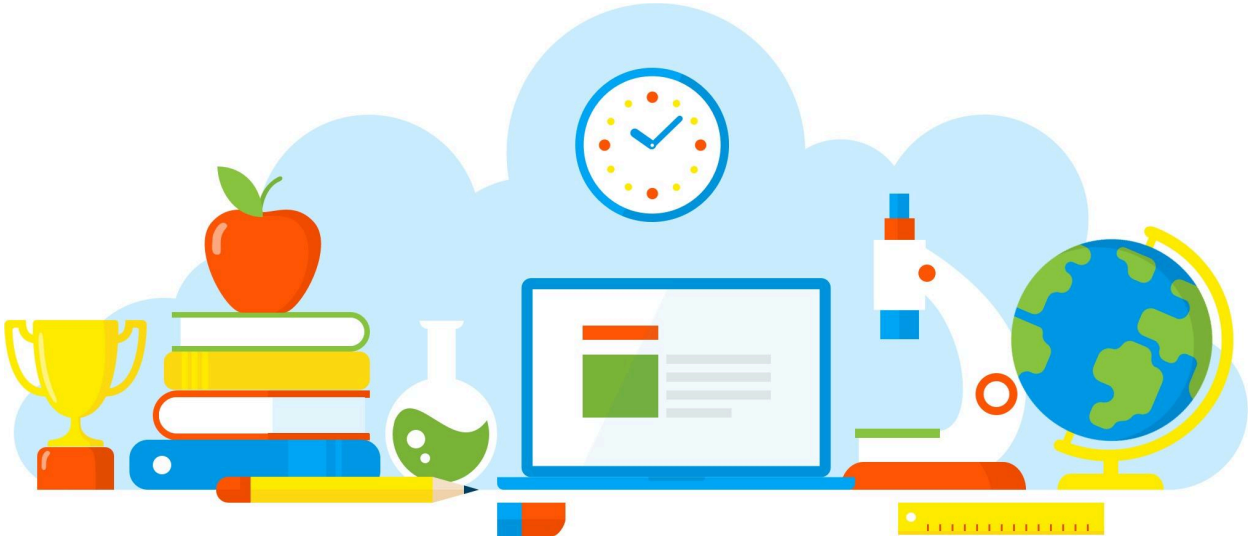
Included

Vanilla Web Development

- html
- CSS
- Javascript

Frameworks

- Vue.js
- Nuxt.js



Part 1
CSS

CSS Cheatsheets

ArSiJa
arsija.net

1. CSS Selectors

```
p { color: blue; }              /* element selector */
#id { color: red; }             /* id selector */
.class { color: green; }        /* class selector */
* { margin: 0; }                /* universal selector */

div p { color: purple; }        /* descendant */
div > p { color: pink; }        /* direct child */
div + p { color: orange; }      /* adjacent sibling */
div ~ p { color: teal; }        /* general sibling */

input[type="text"] { color: blue; }
a[target="_blank"] { color: red; }

ul li:first-child { font-weight: bold; }
ul li:last-child { font-style: italic; }
ul li:nth-child(2) { color: orange; }
```

2. Colors

```
color: red;
color: #ff0000;
color: rgb(255,0,0);
color: rgba(255,0,0,0.5);
color: hsl(0,100%,50%);
opacity: 0.8;
```

3. Text

```
font-family: Arial, sans-serif;
font-size: 16px;
font-weight: bold;
font-style: italic;
text-align: center;
text-decoration: none;
text-transform: uppercase;
line-height: 1.5;
letter-spacing: 2px;
word-spacing: 5px;
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
```

4. Box Model

```
width: 200px;
height: 100px;
max-width: 100%;
min-height: 50px;
margin: 10px auto;
padding: 20px;
border: 2px solid black;
border-radius: 10px;
box-shadow: 2px 2px 10px rgba(0,0,0,0.2);
box-sizing: border-box;
```

5. Background

```
background-color: lightblue;
background-image: url("image.jpg");
background-repeat: no-repeat;
background-size: cover;
background-position: center;
background-attachment: fixed;
background-clip: content-box;
background-origin: border-box;

background: linear-gradient(to right, red, yellow);
background: radial-gradient(circle, red, yellow, green);
```

6. Display & Positioning

```
display: block;
display: inline;
display: inline-block;
display: none;

visibility: hidden;

position: static;
position: relative;
position: absolute;
position: fixed;
position: sticky;

float: left;
clear: both;

z-index: 100;

overflow: hidden;
overflow-x: auto;
overflow-y: scroll;
```

7. Flexbox

```
display: flex;
flex-direction: row;
flex-direction: column-reverse;

justify-content: center;
justify-content: space-between;

align-items: center;
align-self: flex-start;

gap: 10px;
flex-wrap: wrap;

flex: 1;
```

8. Grid

```
display: grid;
grid-template-columns: repeat(3, 1fr);
grid-template-rows: auto;
grid-auto-rows: minmax(100px, auto);

gap: 10px;

grid-column: 1 / 3;
grid-row: 1 / 2;

align-items: start;
justify-items: center;
place-items: center;
```

9. Transitions & Animations

```
div {
  transition: all 0.3s ease-in-out;
}
div:hover {
  transform: scale(1.1) rotate(5deg);
}

@keyframes bounce {
  0% { transform: translateY(0); }
  50% { transform: translateY(-20px); }
  100% { transform: translateY(0); }
}

div {
  animation: bounce 1s infinite alternate;
}

animation-delay: 0.5s;
animation-iteration-count: infinite;
animation-timing-function: ease-in-out;
```

10. Common Units

```
px → pixels
% → relative to parent
em → relative to parent font size
rem → relative to root font size
vh / vw → viewport height / width
fr → grid fraction unit
ch → width of "0" character
ex → height of lowercase "x"
vmin / vmax → relative to viewport smallest/largest dimension
```

11. Pseudo-classes & Pseudo-elements

```
a:hover { color: red; }
a:active { color: green; }
a:visited { color: purple; }

p::first-letter { font-size: 24px; }
p::before { content: " "; }
p::after { content: " "; }

input:focus { border: 2px solid blue; }
input:disabled { background: #ccc; }

:root { --main-color: blue; }
```

12. CSS Variables

```
:root {
  --main-color: #3498db;
  --padding: 10px;
}
div {
  color: var(--main-color);
  padding: var(--padding);
}
```

13. Transformations

```
transform: translate(50px, 100px);
transform: rotate(45deg);
transform: scale(1.5);
transform: skew(20deg, 10deg);
transform-origin: center;
```

14. Filters

```
filter: blur(5px);
filter: brightness(120%);
filter: contrast(150%);
filter: grayscale(100%);
filter: hue-rotate(90deg);
filter: invert(100%);
filter: opacity(50%);
filter: saturate(200%);
filter: sepia(100%);
```

Part 2
html

HTML Cheatsheets

ArSiJa
arsija.net

THE COMPLETE HTML CHEAT SHEET

Powered by HOSTINGER

www.hostinger.com

Table of Content

Hostinger Tutorials

For web developers, it is crucial to be proficient in HTML. And while HTML is not the most difficult to get accustomed to, one can still manage to forget all the nooks and crannies it has to offer. A good solution, therefore, is to always have a cheat sheet at hand, helping you in your most troubling moments.

Document Summary

Let us see how we can break the code up in different components:

<html> ... </html>
This tag specifies that the webpage is written in HTML. It appears at the very first and last line of the webpage. It is mainly used to show that the page uses HTML5 – the latest version of the language. Also known as the root element, this tag can be thought of as a parent tag for every other tag used in the page.

<head> ... </head>
This tag is used to specify meta data about the webpage. It includes the webpage's name, its dependencies (JS and CSS scripts), font usage etc.

<title> ... </title>
As the name suggests, this tag contains the title/name of the webpage. You can see this in your browser's title bar for every webpage open in the browser. Search engines use this tag to extract the topic of the webpage, which is quite convenient when ranking relevant search results.

<body> ... </body>
Everything the user sees on a webpage is written inside this tag. It is a container for all the contents of the webpage.

Example

```
<html>
    <head>
        <title>My First Website</title>
    </head>
    <body>

    </body>
</html>
```

Document Information

<base/>
Used to specify the base URL of your site, this tag makes linking to internal links on your site cleaner.

<meta/>
This is the meta data tag for the webpage. Can be useful for mentioning the page's author, keywords, original published date etc.

<link/>
This is used to link to scripts external to the webpage. Typically utilized for including stylesheets.

<style> ... </style>
The style tag can be used as an alternative to an external style sheet, or complement it. Includes the webpage's appearance information.

<script> ... </script>
Used to add code snippets, typically in JavaScript, to make webpage dynamic. It can also be used to just link to an external script.

Example

```
<html>
    <head>
        <meta charset="utf-8">
        <base href="http://myfirstwebsite.com"
        target="_blank" />
        <title>My Beautiful Website</title>
        <link rel="stylesheet" href="/css/master.css">
        <script type="text/javascript">
        var dummy = 0;
        </script>
    </head>
    <body>

    </body>
</html>
```

Document Structure

<h1..h6> ... </h1..h6>
Six different variations of writing a heading. <h1> has the largest font size, while <h6> has the smallest.

** ... **
A webpage's content is usually divided into blocks, specified by the div tag.

** ... **
This tag injects inline elements, like an image, icon, emoticon without ruining the formatting / styling of the page.

** ... **
Plain text is placed inside this tag.

**
**
A line break for webpages. Is used when wanting to write a new line.

<hr/>
Similar to the above tag. But in addition to switching to the next line, this tag also draws a horizontal bar to indicate the end of the section.

Example

```
<div>
    <h1>Top 5 Greatest Films</h1>
    <p>These are considered the greatest
    <span>reel-icon</span> of all time </p>
    <hr/>
    <h2>1. The Godfather</h2>
    <p>This 1972 classic stars Marlon Brando and
    Al Pacino.</p>
</div>
```

Text Formatting

** ... **
Makes text bold. Used to emphasize a point

** ... **
Alternative to the above tag, also creates bold text.

** ... **
Another emphasis tag, but this displays text in italics.

** ... **
Also used to display text in italics, but does not emphasize it like the above tag.

<tt> ... </tt>
Formatting for typewriter-like text. No longer supported in HTML5.

<strike> ... </strike>
Another old tag, this is used to draw a line at the center of the text, so as to make it appear unimportant or no longer useful.

<cite> ... </cite>
Tag for citing author of a quote.

<del> ... </del>
Pre-formatted, 'monospace' text laid out with whitespace inside the element intact.

<ins> ... </ins>
Denotes text that has been inserted into the webpage.

<blockquote> ... </blockquote>
Quotes often go into this tag. Is used in tandem with the <cite> tag.

<q> ... </q>
Similar to the above tag, but for shorter quotes.

<abbr> ... </abbr>
Denotes abbreviations, along with the full forms.

<acronym> ... </acronym>
Tag for acronyms. No HTML5 support.

<address> ... </address>
Tag for specifying author's contact details.

<dfn> ... </dfn>
Tag dedicated for definitions.

<code> ... </code>
This is used to display code snippets within a paragraph.

_{...}
Used for writing a subscript (smaller font just below the mid-point of normal font). Example: ax

^{...}
Similar to the above tag, but for superscripting.

<small> ... </small>
Reduces text size. In HTML5, it often refers to redundant or invalid information.

Example

```
<p><strong>Bold text</strong> Regular text
<em>some words in italics</em> regular text
once again.</p>

<blockquote>
Anyone who has never made a mistake has never
tried anything new.<cite>- Albert Einstein</cite>
</blockquote>

<pre>
    Some pre-formatted text
</pre>
<p>A code snippet: <code>some code</code></p>
```

Links

** ... **
Anchor tag. Primarily used for including hyperlinks.

** ... **
Tag dedicated to sending emails.

** ... **
Anchor tag for mentioning contact numbers. As the numbers are clickable, this can be particularly beneficial for mobile users.

** ... **
This tag can be used to quickly navigate to a different part of the webpage.

** ... **
A variation of the above tag, this is only meant to navigate to a div section of the webpage.

Images

<img />
A tag to display images in the webpage.

src="url"
The URL or path where the image is located on your drive or on the web.

alt="text"
The text written here is displayed when user hovers mouse over the image. Can be used to give additional details of the image.

height=""
Specifies image height in pixels or percentages.

width=""
Specifies image width in pixels or percentages.

align=""
The relative alignment of the image. Can change with changes to other elements in the webpage.

border=""
Specifies border thickness of the image. If not mentioned, defaults to 0.

<map> ... </map>
Denotes an interactive (clickable) image.

<map name=""> ... </map>
Name of the map associated between the image and the map.

<area />
Specifies image map area.

shape=""
Shape of the area.

coords=""
Coordinates of the vital information of the shape. Example: vertices for rectangles, center/radius for circles.

Example

```
<img src="planets.gif" width="145" height="126"
alt="Planets" usemap="#planetmap">
<map name="planetmap">
    <area shape="rect" coords="0,0,60,100"
    href="sun.htm" alt="Sun">
    <area shape="circle" coords="90,58,3"
    href="mercur.htm" alt="Mercury">
    <area shape="circle" coords="124,58,8"
    href="venus.htm" alt="Venus">
</map>
```

Lists

** ... **
Tag for ordered or numbered list of items.

** ... **
Contrary to the above tag, used for unordered list of items.

** ... **
Individual item as part of a list.

<dl> ... </dl>
Tag for list of items with definitions.

<dt> ... </dt>
The definition of a single term inline with body content.

<dd> ... </dd>
The description for the defined term.

Example

```
<ol>
    <li>Monday</li>
    <li>Tuesday</li>
    <li>Wednesday</li>
</ol>
<ul>
    <li>France</li>
    <li>Germany</li>
    <li>Italy</li>
</ul>
<dl>
    <dt>Toyota</dt>
    <dd>Japanese car brand</dd>
    <dt>Armani</dt>
    <dd>Italian fashion brand</dd>
</dl>
```

Forms

<form> ... </form>
The parent tag for an HTML form.

action="url"
The URL listed here is where the form data will be submitted once user fills it.

method=""
It specifies which HTTP method (POST or GET) would be used to submit the form.

enctype=""
Only for POST method, this dictates the data encoding scheme to be used when form is submitted.

autocomplete
Determines if the form has auto-complete enabled.

novalidate
Determines whether the form should be validated before submission.

accept-charsets
Determines character encodings when form is submitted.

target
After submission, the form response is displayed wherever this refers to, usually has the following values: _blank, _self, _parent, _top

<fieldset> ... </fieldset>
Identifies the group of all fields on the form.

<label> ... </label>
This is used to label a field in the form.

<legend> ... </legend>
This operates as a caption for the <fieldset> element.

<input />
This tag is used to take input from the user. Input type is determined by a number of attributes.

Input Type Attributes

type=""
Determines which type of input (text, dates, password) is requested from the user.

name=""
Specifies the name of the input field.

value=""
Specifies the value contained currently in the input field.

size=""
Determines the input element width (number of characters).

maxlength=""
Specifies the most input field characters allowed.

required
Makes an input field compulsory to be filled by the user. The form cannot be submitted if a required field is left empty.

width=""
Determines the width of the input element, in pixel values.

height=""
Determines the height of the input element, in pixel values.

placeholder=""
Can be used to give hints to the user about the nature of the requested data.

pattern=""
Specifies a regular expression, which can be used to look for patterns in the user's text.

min=""
The minimum value allowed for an <input> element.

max=""
The maximum value allowed for an <input> element.

autofocus
Forces focus on the input element when webpage loads completely.

disabled
Disables the input element. User can no longer enter data.

<textarea> ... </textarea>
For longer strings of input. Can be used to get multi-sentence text from the user.

<select> ... </select>
This tag specifies a list of options which the user can choose from.

Select Attributes

name=""
The name of a particular list of options.

size=""
Total number of options given to the user.

multiple
States whether the user can choose multiple options from the list.

required
Specifies whether choosing an option/s is necessary for form submission.

autofocus
Specifies that a drop-down list automatically comes into focus after a page loads.

<option> ... </option>
Tag for listing individual items in the list of options.

Option Attributes

value=""
The text visible to the user for any given option.

selected
Determines which option is selected by default when the form loads.

<button> ... </button>
Tag for creating a button for form submission.

Example

```
<form action="form_submit.php" method="post">
    <fieldset>
        <legend>Bio:</legend>
        First name:<br>
        <input type="text" name="first-name"
        value="John" placeholder="Please
        enter your first name here"><br>
        Last name:<br>
        <input type="text" name="last-name"
        value="Doe" placeholder="Please
        enter your last name here"><br><br>
        Favorite sport:<br>
        <select>
            <option value="soccer">Soccer
            </option>
            <option value="tennis">Tennis
            </option>
            <option value="golf">Golf
            </option>
        </select>
        <textarea name="description">
        </textarea>
        <input type="submit" value="Submit">
    </fieldset>
</form>
```

Tables

<table> ... </table>
Marks a table in a webpage.

<caption> ... </caption>
Description of the table is placed inside this tag.

<thead> ... </thead>
Specifies information pertaining to specific columns of the table.

<tbody> ... </tbody>
The body of a table, where the data is held.

<tfoot> ... </tfoot>
Determines the footer of the table.

<tr> ... </tr>
Denotes a single row in a table.

<th> ... </th>
The value of a heading of a table's column.

<td> ... </td>
A single cell of a table. Contains the actual value/data.

<colgroup> ... </colgroup>
Used for grouping columns together.

<col>
Denotes a column inside a table.

Example

```
<table>
    <colgroup>
        <col span="2">
        <col>
    </colgroup>
    <tr>
        <th>Name</th>
        <th>Major</th>
        <th>GPA</th>
    </tr>
    <tr>
        <td>Bob</td>
        <td>Law</td>
        <td>3.55</td>
    </tr>
    <tr>
        <td>Alice</td>
        <td>Medicine</td>
        <td>3.61</td>
    </tr>
</table>
```

Objects and iFrames

<object> ... </object>
This tag is used to embed additional multimedia into a webpage. Can be audio, video, document (pdf) etc.

height=""
Determines object height in pixel values.

width=""
Determines object width in pixel values.

type=""
The type/format of the object's contents.

<iframe> ... </iframe>
An inline block of content, this is used as a container for multimedia in a flexible manner. It floats inside a webpage, meaning it is placed relative to other webpage items.

iFrame Attributes

name=""
The name of the iFrame.

src=""
The source URL/path of the multimedia object to be held inside the iFrame.

srcdoc=""
Any HTML content to be displayed inside the iFrame.

height=""
Determines the height of the iFrame.

width=" "
Determines the width of the iFrame.

<param />
For iFrame customization. This includes additional parameters to go along with the content.

<embed> ... </embed>
This is used to embed external objects, like plugins (e.g. a flash video).

Embed Attributes

height=""
Determines the height of the embedded item.

width=""
Determines the width of the embedded item.

type=""
The type or format of the embedded content.

src=""
The URL/path of the embedded item.

Example

```
<object width="1000" height="1000"></object>
<iframe src="some_other_webpage.html"
width="500" height="500"></iframe>
<embed src="some_video.swf" width="500"
height="500"></embed>
```

HTML5 New Tags

<header> ... </header>
Specifies the webpage header. Could also be used for objects inside the webpage.

<footer> ... </footer>
Specifies the webpage footer. Could also be used for objects inside the webpage.

<main>...</main>
Marks the main content of the webpage.

<article>...</article>
Denotes an article.

<aside> ... </aside>
Denotes content displayed in a sidebar of the webpage.

<section>...</section>
Specifies a particular section in the webpage.

<details> ... </details>
Used for additional information. User has the option to view or hide this.

<summary> ... </summary>
Used as a heading for the above tag. Is always visible to the user.

<dialog>...</dialog>
Used to create a dialog box.

<figure>...</figure>
A tag reserved for figures (diagrams, charts) in HTML5.

<figcaption> ... </figcaption>
A description of the figure is placed inside these.

<mark>...</mark>
Used to highlight a particular portion of the text.

<nav>...</nav>
Navigation links for the user in a webpage.

<menuitem>...</menuitem>
A particular item from a list or a menu.

<meter>...</meter>
Measures data within a given range.

<progress>...</progress>
Typically used as a progress bar, this is used to track progress.

<rp>...</rp>
This tag is meant for showing text for browsers without ruby annotation support.

<rt>...</rt>
Displays East Asian typography character details.

<ruby>...</ruby>
Describes a Ruby annotation for East Asian typography

<time>...</time>
Tag for formatting date and time.

<wbr>
A line-break within the content.

Collective Character Objects

" "
Quotation Marks - "

< <
Less than sign - <

> >
Greater than sign - >

**   **
Non-breaking space

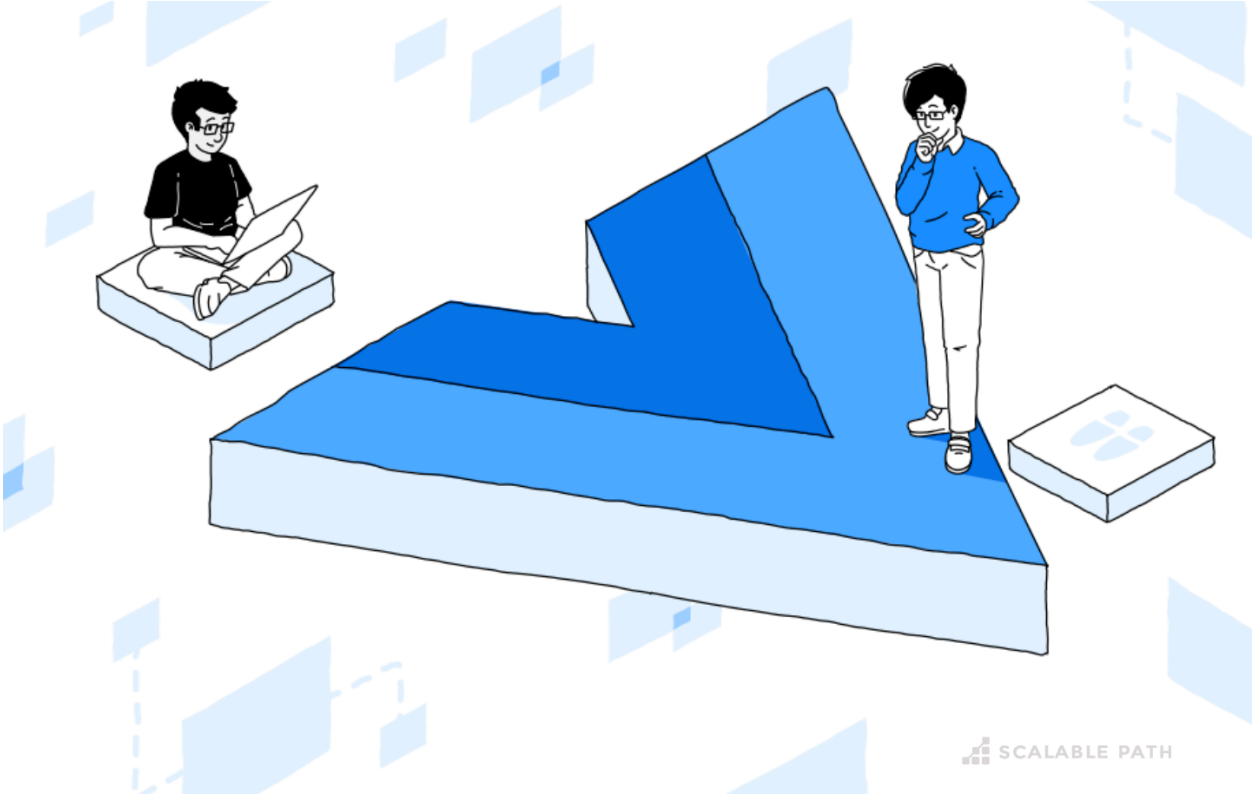
© ©
Copyright symbol - ©

& &
Ampersand - &

@ Ü
@ Symbol - @

• ö
Small bullet - •

™ û
Trademark symbol - ™



Part 3
Vue

Vue Cheatsheets

ArSiJa
arsija.net

VUE 3 CHEATSHEET FOR DEVELOPERS

Put together by your friends at learnvue.co

CREATING YOUR APP WITH VITE

Quick Vue3 development environment

```
npm init vite-app <project-name>
cd <project-name>
npm install
npm run dev
```

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

```
const count = (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>
```

VUE 3 CHEATSHEET FOR DEVELOPERS

Put together by your friends at learnvue.co

Loop Values in Object

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

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)

```
context.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>
```

VUE 3 CHEATSHEET FOR DEVELOPERS

Put together by your friends at learnvue.co

Parent Has Access to MyButton post data

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

DYNAMIC COMPONENTS

Changes the rendered component - finds a registered component with the given name.

```
<component :is='componentName'/>
```

KEEP-ALIVE ELEMENTS

Stores a cached version of dynamic components when not visible. Avoids having to create a new component whenever toggled.

```
<keep-alive>
  <component :is='componentName'/>
</keep-alive>
```

COMPOSITION API

Everything returned by setup() is exposed to the template.

```
import { ref, reactive } from 'vue'
export default {
  setup(props, context) {
    const val = ref('example')
    const obj = reactive({ count: 0 })

    const evtHandler = () => {/*...*/}

    return {
       val, obj, evtHandler
    }
  }
}
```

SETUP() CONTEXT OBJECT PROPERTIES

attrs	Has component's attributes
slots	Has component's slots
emit	Function to emit events

VUEJS LIFECYCLE HOOKS

*beforeCreate	Use setup() instead
*created	Use setup() instead
onBeforeMount	Before mounting DOM
onMounted	DOM can be accessed
onBeforeUpdate	Reactive data changes
onUpdated	DOM has been updated
onBeforeUnmount	Component still complete
onUnmounted	Teardown complete

EXAMPLE LIFECYCLE HOOK CODE

```
import { onMounted } from 'vue'
// ...
setup() {
  onMounted(() => {
    console.log('component mounted!')
  }
}
```

VUE GLOBAL METHODS

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

VUE 3 CHEATSHEET FOR DEVELOPERS

Put together by your friends at learnvue.co

COMPUTED PROPERTIES

A computed property is a value that is calculated using one or more other properties.

```
setup() {
  const a = ref(1)
  const b = computed(() => a.value * 2)

  return { a, b }
}
```

WATCHEFFECT()

Listens to reactive dependencies and runs a method when one changes. Also runs on init.

```
setup() {
  const site = ref('learnvue.co')

  watchEffect(() => {
    console.log(site.value)
  })

  return { site }
}
```

TEMPLATE REFS

Give access to DOM elements.

```
// template
<div ref='example'> Example Div </div>
// script
setup() {
   const example = ref('learnvue.co')
   // wait for DOM to mount
   onMounted(() => {
     console.log(example.value)
   })

   return { example }
}
```

VUE OBJECT API OPTIONS

If you decide not to use the Composition API, your components will look similar to Vue2 with the Options API.

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

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	Bulma
Fish-UI	Muse UI	Vue Blu

CONTACT

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

Hope this helped!

Part 4

Nuxt

Nuxt Cheatsheets

ArSiJa
arsija.net

Vue Mastery

NUXT CHEAT SHEET 1/2

Pages

use NuxtPage in app.vue to render the page

app.vue

```
<template>
  <NuxtPage />
</template>
```

pages/index.vue

```
<template>
  <h1>Home</h1>
  ...
</template>
```

localhost:3000/

Home
...

index.vue is the root route by default

pages/about.vue

```
<template>
  <h1>About</h1>
  ...
</template>
```

localhost:3000/about

About
...

Pages with Layout

layouts/default.vue

```
<template>
  <div>
    <nav>
      <NuxtLink to="/">Home</NuxtLink>
      <NuxtLink to="/about">About</NuxtLink>
    </nav>
    <main>
      <slot />
    </main>
  </div>
</template>
```

the page content will appear here

app.vue

```
<template>
  <NuxtLayout>
    <NuxtPage />
  </NuxtLayout>
</template>
```

pages/index.vue

```
<template>
  <h1>Home</h1>
  ...
</template>
```

use NuxtLayout and NuxtPage in app.vue in app.vue to connect everything together

localhost:3000/

Home | About

Home
...

Routing

[foo].vue

Dynamic Route
e.g. /about
`params: {foo: "about"}`

my-[foo].vue

Mixed Route
e.g. /my-inventory
`params: {foo: "inventory"}`

[[foo]].vue

Optional Route
e.g. /
`params: {foo: ""}`

[...foo].vue

Catch-All Route
e.g. /profile/image
`params: {foo: ["profile","image"]}`

Access Route Params

```
<script setup>
const params = useRoute().params
</script>
```

no need to import useRoute

Vue Mastery

NUXT CHEAT SHEET 2/2

SSR-Compatible Shared State

any composable in this folder gets imported to the components automatically

composables/useWidth.js

```
export function useWidth() {
  return useState('width', () => {
    return 1000
  })
}
```

this function passed to useState will only get executed on the server

pages/index.vue

```
<script setup>
const width = useWidth()
console.log(width.value)
</script>
```

no need to import

pages/about.vue

```
<script setup>
const width = useWidth()
console.log(width.value)
</script>
```

same state

useFetch & API

pages/index.vue

```
<script setup>
const { data } = await useFetch('/api/foo')
</script>
```

use useFetch if you're fetching in a component

fetch from an API file with the same name on the server

server/api/foo.js

```
export default defineEventHandler(async (event) => {
  return [ 1, 2, 3 ]
})
```

remember to use async if you're using await in the function

the event param provides the data of the request

Rendering Mode Config

nuxt.config.js

```
export default defineNuxtConfig({
  routeRules: {
    '/page-name': { ssr: true }
  }
})
```

route path

Rendering Mode (Default) *Server-Side Rendering*

Rendering Mode *Client-Side Only*

```
{ ssr: false }
```

Rendering Mode *Static Generation*

```
{ prerender: true }
```

Rendering Mode *Stale-While-Revalidate*

```
{ swr: true }
```

Rendering Mode *Incremental Static Regeneration*

```
{ isr: true }
```

NUXT 3 ESSENTIALS CHEAT SHEET

Insidewebdev

STARTING A NEW PROJECT

Create a project using nuxi:

```
$ npx nuxi init <project-name>
$ cd <project-name>
$ npm install        Installs dependencies
$ npm run dev        Runs the project
```

FOLDER STRUCTURE

- **ASSETS -** Uncompiled assets (like Sass, fonts Images)
- **COMPONENTS -** Components are automatically imported based on the folder and file name
- **COMPOSABLES -** Directory to automatically import your composables into your application
- **CONTENT -** Contains files to create a file based CMS
- **LAYOUTS -** Application layouts
- **MIDDLEWARE -** Custom functions to run before each page
- **PAGES -** Application views & routes from which the router is dynamically generated
- **PLUGINS -** JS plugins run before Vue.js init
- **SERVER -** Create serverless functions

STATE MANAGEMENT

Nuxt provides `useState` composable to create a reactive and SSR-friendly shared state across components.

🚨 Never define **const state = ref()** outside of **<script setup>** or **setup()** function.

✅ Instead use **const useX = () => useState('x')**

Basic

```
<script setup>
const counter = useState('counter', () =>  Math.random() * 1000)
</script>
```

Shared state

Composables defined inside ***~/composables*** folder are auto-imported and with that we can share states and import them across the app.

composables/states.ts

```
export const useCounter = () => useState<number>('counter', () => 0)
export const useColor = () => useState<string>('color', () => 'pink')
```

app.vue

```
<script setup>
const color = useColor() // Same as useState('color')
</script>

<template>
  <p>Current color: {{ color }}</p>
</template>
```

PAGES

Nuxt generates each app route from the folder `pages` directory.

```
pages
  index.vue              loads the root path /
  posts
    [...slug].vue        [...x] catch all route /posts/my-slug
    index.vue            /posts
  users-[group]          we can also add params inside a folder . /users-customer
    [id].vue             [ ] defines a dynamic route with a param. /users-admin/123
```

Access directly inside the template

pages/users-[group]/[id].vue

```
<template>
  <p>{{ $route.params.group }} - {{ $route.params.id }}</p>
</template>
```

Access inside script tag using Composition API

pages/users-[group]/[id].vue

```
<script setup>
const route = useRoute()

const { group, id } = route.params

</script>
```

DATA FETCHING

Nuxt provides **useFetch**, **useLazyFetch**, **useAsyncData** and **useLazyAsyncData** to handle data fetching within your application.

⚠️ **useFetch**, **useLazyFetch**, **useAsyncData** and **useLazyAsyncData** only work during setup or Lifecycle Hooks

useFetch()

```
<script setup>
const { data: count, pending, refresh, error  } = await useFetch('/api/count')
</script>

<template>
  Page visits: {{ count }}
</template>
```

useAsyncData()

```
<script setup>
const { data } = await useAsyncData('count', () => $fetch('/api/count'))
</script>

<template>
  Page visits: {{ data }}
</template>
```

👉 Difference between useFetch and useAsyncData:
useFetch receives a URL and gets that data, whereas **useAsyncData** might have more complex logic.
useFetch(url) is nearly equivalent to ***useAsyncData(url, () => $fetch(url))***

useLazyFetch() and useLazyAsyncData()

These composables behave identically to ***useFetch*** and ***useAsyncData*** with the ***lazy: true*** option set. In other words, the async function does not block navigation.

NUXT 3 ESSENTIALS CHEAT SHEET

Insidewebdev

SERVER ROUTES

Files inside the ***~/server/api*** are automatically prefixed with ***/api*** in their route. For adding server routes without ***/api*** prefix, you can instead put them into ***~/server/routes*** directory.

server/api/route.post.js

```
import { sendError } from "h3"

export default defineEventHandler(async (event) => {
    const config = useRuntimeConfig()  // Get the runtime config

    const query = useQuery(event) // Get the query from the event
    const body = await useBody(event) // Get the body from the event
    const headers = useHead(event) // Get the headers from the event
    const cookies = useCookies(event) // Get the cookies from the event

    // Throw error
    if(something) {
        return sendError(event, createError({
            statusCode: 400,
            statusMessage: ‘Error message’
        }))
    }

    return {
        // returns a response object
    }
})
```

Matching route params

Server routes can use dynamic parameters within brackets in the file name like ***/api/hello/[name].ts*** and accessed via ***event.context.params***.
Catching all routes its as easy as using the spread operator ***[...name]***

/server/api/hello/[name].ts

```
export default defineEventHandler(event => `Hello, ${event.context.params.name}!`)
```

Matching HTTP Method

Handle file names can be suffixed with ***.get***, ***.post***, ***.put***, ***.delete***, ... to match request’s.

/server/api/test.**get**.ts

```
export default defineEventHandler(() => 'Test get handler')
```

/server/api/test.**post**.ts

```
export default defineEventHandler(() => 'Test post handler')
```

SERVER MIDDLEWARE

Nuxt will automatically read in any file in the ***~/server/middleware*** to create server middleware for your project.

server/middleware/auth.js

```
export default defineEventHandler((event) => {
  console.log('New request: ' + event.req.url)
})
```

TELEPORT

```
<template>
  <button @click="open = true">
    Open Modal
  </button>
  <Teleport to="body">
    <div v-if="open" class="modal">
      <p>Hello from the modal!</p>
      <button @click="open = false">
        Close
      </button>
    </div>
  </Teleport>
</template>
```

RUNTIME CONFIG

Expose runtime config

nuxt.config.ts

```
export default defineNuxtConfig({
  runtimeConfig: {
    apiSecret: process.env.API_SECRET,
    public: {
      apiBase: process.env.API_BASE,
    }
  },
})
```

Accessing runtime config

Client

```
<script setup>
const config = useRuntimeConfig()
console.log('Runtime config:', config)
if (process.server) {
  console.log('API secret:',
config.apiSecret)
}
</script>
```

Server

```
export default defineEventHandler(async
(event) => {
    const config = useRuntimeConfig()

    return {
        // returns a response object
    }
})
```

For more quality programming courses

insidewebdev

ArSiJa

Switzerland
silvan@arsija.net
www.arsija.net

Web Development Cheatsheet V1

26. January 2026

About

A cheatsheet with all important information for Web Development, mainly for Vue/Nuxt.

Goals

1. Create an all in one Document for Web Development
2. Make it Organized and clear.

Included

Vanilla Web Development

- html
- CSS
- Javascript

Frameworks

1. Vue.js
2. Nuxt.js