Customize the theme with personal styles

A personal website should look different to others to make it unique or standout. This post guides to add some must-have elements of a blog like tag cloud, tag page, list of posts into the theme's template. This also add some styles and scripts to the pages in order to get a simple look which goes straight to the content but still have clear and beautiful visual for reading not only on screens but also in papers.

#jinja #css #javascript

Last update: 2021-08-10 16:50:49

Table of Content

1. The post's title

By using the Awesome Pages plugin and the Section index plugin, the navigation sidebar can show a good structure of posts. However, each entry is displaying the post's title with long text, this make the navigation bar look a bit messy.

It is easy to add a full title into a post by using the Metadata section. For example:

```
---
title: Customize theme
title_full: Customize the theme with personal styles
description: A personal website should look different to others ...
---
```

However, the content of each post will also display the short title. To fix this, I am going to override the main template. Before do it, I will add tags in the Metadata section also.

2. The post's tags

A tag is a word or a phrase that describes one main point of a blog post's content. They are an easy way to attach labels to the content and link similar posts together.

The tags of a post are defined in the Metadata section, then they are gathered and processed in templates later.

```
---
title: title
description: description
tags:
    - python
    - mkdocs
---
```

3. The Tags page

The tag page is the place to list all tags, and list all pages that have a common tag. A new page will be created at `docs\tags\index.md`. There is a method to use MkDocs Macros in Markdown template, but it is quite complicated.

I use Jinja syntax to create the content of the Tags page, therefore, create a new file at `overrides\tags.html` and use it as the template for the Tags page:

docs\tags\index.md

```
---
title: Tags
description: Tags and list of pages
```

```
template: tags.html
hide:
    - disqus
---

# Tags
```

The `tags.html` template to include 2 parts:

- `tag-cloud.html` : make a tag cloud to see how many pages are associated with a tag

- `tag-list-pages.html` : for each tag, list all pages having that tag to show similar articles together

overrides\tags.html

```
{% extends "main.html" %}

{% block page_content %}
    <style>
        .tag-cloud-toc {
            display: none;
        }
    </style>
    {% include "partials/tag-cloud.html" %}
    {% include "partials/tag-page-list.html" %}
{% endblock %}
```

Tags will have random colors, to easily disguise them to each other. A helper `random_color()` macro that returns a random color looks like:

overrides\partials\random-colors.html

```
{%- macro random_color() -%}
{{- ["DarkRed",
"DarkGoldenrod",
"DarkGreen",
"DarkOliveGreen",
"DarkCyan",
"DarkTurquoise",
"DarkBlue",
"DarkMagenta",
"DarkViolet",
"DarkSlateBlue",
"DarkOrchid",
"DarkSlateGray"] | random -}}
{%- endmacro -%}
```

Then it can be imported as:

```
{% from "partials/random-colors.html" import random_color %}
```

3.1. The tag cloud

The tag cloud shows all tags in different size and color. The bigger a tag is, the more pages mention that tag. Steps to make a tag cloud:

1. Scan all pages and create a list of pairs `(tag, pages[])`

```
{% set tags=[] %}
{# scan all pages #}
{% for p in pages %}
{% if p.page.meta.tags %}
  {# extract tags if available #}
  {% for tag in p.page.meta.tags %}
  {% if tags|length %}
    {% set ns = namespace(found=False) %}
    {# read more about scope at
    https://jinja.palletsprojects.com/en/2.11.x/templates/#assignments
    #}
    {# check if tag exists, append to its page list #}
    {% for item in tags %}
    {% set t, ps = item %}
    {% if tag == t %}
      {% set ns.found = True %}
      {{ ps.append(p.page) or "" }} {# use (or "") to not print} #}
    {% endif %}
    {% endfor %}
    {# if tag doesn't exist, create new page list#}
    {% if not ns.found %}
    {{ tags.append((tag, [p.page])) or "" }}
    {% endif %}
  {% else %}
    {{ tags.append((tag, [p.page])) or "" }}
  {% endif %}
  {% endfor %}
{% endif %}
{% endfor %}
```

2. Count the number of pages for each tag then show each tag with different text size and color using `font-size` and `color` attributes

```
<p class="md-nav">
    <label class="md-nav__title">Tag cloud</label>
</p>
<div class="tag-cloud-content">
{% if tags|count %}
    {% for item in tags %}
    {% set tag, ps = item %}
    {# create a link with tag name #}
    {# font size is based on the page count #}
    <a class="tag" href="{{ config.site_url }}tags/#{{ tag }}">
        <span class="tag-name" style="
        font-size:{{ 0.6+ps|count*0.1}}rem;
```

```
        color:{{ random_color() }};
        ">
        {{- tag -}}
        </span>
        <sup class="tag-count">{{- ps|count -}}</sup>
    </a>
    {% endfor %}
{% else %}
    <div>
    <h3>No tag found!</h3>
    </div>
{% endif %}
</div>
```

3.2. List pages of a tag

This section is simple as it just needs to loop through the list of pairs `(tag, pages[])` and create a link to each page. Steps to take that:

1. Scan all pages and create a list of pairs `(tag, pages[])`

   > see above section

2. Show each tag with the list of pages in a collapsible `<details>` block

   ```
   <div class="tag-page-list">
   {% for item in tags %}
       {% set tag, ps = item %}
       <details class="note" id={{ tag }}>
       <summary>
           {{- tag }} ({{- ps|count -}})
           <a class="headerlink" href="#{{ tag }}">⚓</a>
       </summary>
       <ol>
           {% for p in ps %}
           <li>
           <a href="{{ page.canonical_url }}">
               {%- if p.meta and p.meta.title_full -%}
               {{- p.meta.title_full -}}
               {%- elif p.meta and p.meta.title -%}
               {{- p.meta.title -}}
               {%- else -%}
               {{- p.title -}}`
               {%- endif -%}
           </a>
           </li>
           {% endfor %}
       </ol>
       </details>
   {% endfor %}
   </div>
   ```

3. Only one tag block is open at a time to easily follow the selected tag. To do this, I added a callback of the `toggle` event on all tag blocks. Whenever a block is opened, this script will close all others

```
[...document.getElementsByTagName("details")].forEach((D, _, A) => {
    D.open = false;
    D.addEventListener(
        "toggle",
        (E) => D.open && A.forEach((d) => d != E.target && (d.open = false))
    );
});
```

4. A tag block can be opened via URL with hash being the selected tag

```
var hash = window.location.hash.substr(1);
if (hash) {
    document.getElementById(hash).open = true;
}
```

Visit the Tags to see the result.

4. The recent blog posts

There should be a page showing the recent posts to help users see what is new and updated. With the Revision Date plugin, it is able to use two new meta-data fields: `git_revision_date_localized`, and `git_creation_date_localized` if the option `enable_creation_date` is `true`.

Create new `index.md` file inside the `blog` folder. When using the Section Index plugin, this index file will be merged to the Blog section, therefore, when user selects the Blog label, there is a list of recent posts will be shown.

This page will use the `blog.html` template in which it scans all posts and check the creation date to make a list of posts. Each post should be displayed in a container and be formatted to show the title, the description (at most 250 character using the `truncate` filter), the creation date, and its tags.

> ⚠ Need to check the page's path to filter blog posts. In my code, I use the `abs_url` and its length to check if a page is in the `blog` directory.

Here is the code to sort all pages in order of creation date, and then filter all blog posts to save into the array `blog_pages` which will be used to generate content.

```
{% set blog_pages=[] %}
{% for p in pages|sort(
        attribute='page.meta.git_creation_date_localized',
        reverse=True
    )
%}
    {% set pg = p.page %}
    {% if pg.abs_url.startswith('/blog/') and pg.abs_url|length > 6 %}
        {{ blog_pages.append(pg) or "" }}
    {% endif %}
{% endfor %}
<div class="pages">
    ... create list from blog_pages ...
</div>
```

4.1. Create the pagination

When the number of posts goes bigger, the recent post list becomes longer. It's time to brake the long list into pages — the user can click on the page number to see its children posts.

This is called "Pagination". How to implement it?

Jinja template has the `slice` `filter` to divide a list into sub-lists. Here, I'd like to have maximum of 10 posts on each page.

```
{% set page_num = (blog_pages|count / 10)|int %}
{% if page_num == 0 %}
    {% set page_num = 1 %}
{% endif %}
<div class="pages">
    {% for pg_group in blog_pages|slice(page_num) %}
    <div class="page" id="page{{ loop.index }}">
        {% for pg in pg_group %}
            <div class="post">
                ... create post layout and content ...
            </div>
        {% endfor %}
    </div>
    {% endfor %}
</div>
```

4.2. Create a post-entry

Each post is wrapped inside a `<div class="post">` and its elements are marked with different classes, such as `post-title`, `post-description`, etc. for applying styles later.

```
<div class="post">
    <h3 class="post-title">
        <a class="link" href="{{ pg.canonical_url }}">{{ title }}</a>
```

```
        </h3>
        <p class="post-description">
            {% if pg.meta.description %}
            {{ pg.meta.description | truncate }}
            {% endif %}
        </p>
        <div class="post-extra">
            <span class="post-timestamp">
                {% if pg.meta and pg.meta.git_revision_date_localized %}
                <span class="post-timestamp-update">
                    Updated: {{ pg.meta.git_revision_date_localized -}}
                </span>
                {% endif %}
            </span>
            <span class="post-tags">
                {% if pg.meta.tags %}
                {% for tag in pg.meta.tags %}
                <a class="tag" href="{{ config.site_url }}tags/#{{tag}}">
                    <span class="tag-name"
                          style="color:{{random_color()}};">
                        #{{ tag }}
                    </span>
                </a>
                {% endfor %}
                {% endif %}
            </span>
        </div>
        <hr />
    </div>
```

Here is a simple style to make each post display necessary basic information:

```
.md-typeset .post:first-of-type h3 {
    margin-top: 0;
}
.md-typeset .post-title {
    margin-bottom: 0;
}
.md-typeset .post-extra {
    color: gray;
}
.md-typeset .post-tags {
    float: right;
}
```

4.3. Create active page

To show the current active page, I use pure CSS and JavaScript. The idea is to use the URL hash to detect which page is activated, such as `#page1` .

```
<div class="center">
    <div class="pagination" id="pagination">
    <a href="#">&laquo;</a>
    {% for pg_group in blog_pages|slice(page_num) %}
        <a class="page-number {% if loop.index==1 %}active{% endif%}"
            href="#page{{ loop.index }}">{{ loop.index }}</a>
    {% endfor %}
    <a href="#">&raquo;</a>
    </div>
</div>
```

Then add some styles to the pagination block and its children links:

CSS Styles:

Use `target` keyword to select the selected *page ID*, then show only the target element.

assets\extra.css

```
.md-typeset .pages > .page:target ~ .page:last-child,
.md-typeset .pages > .page {
    display: none;
}
.md-typeset .pages > :last-child,
.md-typeset .pages > .page:target {
    display: block;
}
```

JavaScript

When the page is loaded, a script will run to get all pagination's links, and then add a callback function for click event, that remove `active` class from last activated element and then assign `active` class to the event's source element. Note that the first page is activated by default when the page is loaded.

assets\extra.js

```
var pagination = document.getElementById("pagination");
var links = pagination.getElementsByClassName("page-number");
if (links.length) {
    for (var i = 0; i < links.length; i++) {
        links[i].addEventListener("click", function () {
            var current = pagination.getElementsByClassName("active");
            console.log(current);
            if (current.length) {
                current[0].className = current[0].className.replace(
                    " active",
                    ""
                );
            }
```

```
            this.className += " active";
        });
    }
    links[0].click();
}
```

5. The main template

The `main.html` file, extending the `base.html` template, will be used for all markdown pages, and it is the starting point to add custom template.

To override it, add the `main.html` file in the `overrides` folder. Here are things I'm going to do to add more content into a blog post:

1. Extract metadata to get `title`, `title_full`, `description`, `tags`, and other information

```
{% set title = config.site_name %}
{% if page and page.meta and page.meta.title_full %}
    {% set title = page.meta.title_full %}
{% elif page and page.meta and page.meta.title %}
    {% set title = page.meta.title %}
{% elif page and page.title %}
    {% set title = page.title %}
{% endif %}

{% set description = config.site_description %}
{% if page and page.meta and page.meta.description %}
    {% set description = page.meta.description %}
{% endif %}

{% if page and page.meta and page.meta.banner %}
    {% set image = page.meta.banner %}
{% endif %}

{% if page and page.meta and page.meta.tags %}
    {% set tags = page.meta.tags %}
{% endif %}
```

2. Add block to use the Open Graph protocol to show the page's information when a user shares a page on a social network

```
{% block htmltitle %}
    <title>{{ title | striptags }} - {{ config.site_name }}</title>
{% endblock %}

{% block extrahead %}
    <meta property="og:type" content="website" />
    <meta property="og:title" content="
        {{- title | striptags ~ ' - ' ~ config.site_name -}}
```

```
        " />
        <meta property="og:description" content="{{ description }}" />
        <meta property="og:url" content="{{ page.canonical_url }}" />
        <meta property="og:image" content="
            {%- if image is defined -%}
                {{ page.canonical_url ~ image }}
            {%- else -%}
                {{ config.site_url ~ 'assets/banner.jpg' }}
            {%- endif -%}
            " />
        <meta property="og:site_name" content="{{ config.site_name }}" />
        <meta name="twitter:card" content="summary" />
    {% endblock %}
```

3. The page content should have a `cover` section which displays the `title`, `description` and `tags` on all pages, except the home page.

```
{% block content %}
    {% if not page.is_homepage %}
    <div class="cover">
        <h1 class="page-title">
            {{ title | d(config.site_name, true) }}
        </h1>
        <p class="page-description">{{ description }}</p>
        {% if tags is defined %}
        <p class="page-tags">
            {% for tag in tags %}
            <a class="tag" href="{{ config.site_url }}tags/#{{tag}}">
                <span class="tag-name">
                    #{{ tag }}
                </span>
            </a>
            {% endfor %}
        </p>
        {% endif %}
    </div>
    {% endif %}
    {{ page.content }}
{% endblock %}
```

4. The tag cloud should show in the sidebars based on the page's width

```
{% block site_nav %}
    {% if nav %}
        {% if page and page.meta and page.meta.hide %}
          {% set hidden = "hidden" if "navigation" in page.meta.hide %}
        {% endif %}
        <div class="md-sidebar md-sidebar--primary"
            data-md-component="sidebar"
            data-md-type="navigation" {{ hidden }}>
            <div class="md-sidebar__scrollwrap">
                <div class="md-sidebar__inner">
```

```
                        {% include "partials/nav.html" %}
                        <div class="tag-cloud-nav">
                            {% include "partials/tag-cloud.html" %}
                        </div>
                    </div>
                </div>
            </div>
        {% endif %}
        {% if page.toc and not "toc.integrate" in features %}
            {% if page and page.meta and page.meta.hide %}
                {% set hidden = "hidden" if "toc" in page.meta.hide %}
            {% endif %}
            <div class="md-sidebar md-sidebar--secondary"
                data-md-component="sidebar"
                data-md-type="toc" {{ hidden }}>
                <div class="md-sidebar__scrollwrap">
                    <div class="md-sidebar__inner">
                        {% include "partials/toc.html" %}
                        <div class="tag-cloud-toc">
                            {% include "partials/tag-cloud.html" %}
                        </div>
                    </div>
                </div>
            </div>
        {% endif %}
    {% endblock %}
```

6. Zoom-in Images

As mentioned in the Images section, view-bigimg library helps to zoom and pan images. It's useful when the image is in high resolution and resized to fit site's width.

Download `view-bigimg.css` and `view-bigimg.js` files from the view-bigimg repo, then add them into the addition assets configs in `mkdocs.yml`:

mkdocs.yml

```
extra_css:
    - assets/view-bigimg.css
extra_javascript:
    - assets/view-bigimg.js
```

When click on the image, this library will create a new layer and show the image in a bigger size. However, it must be clicked on the close button to go back to the page's content. I want to simplify this step by just click on the image. Panning still is activated by press and hold. Therefore, I write a function to detect `mousedown` and `mousemove` event, then only close the image if it is a simple click:

assets\extra.js

```
var dragged = false;
document.addEventListener("mousedown", () => (dragged = false));
document.addEventListener("mousemove", () => (dragged = true));

var viewer = new ViewBigimg();
var figures = document.querySelectorAll("img");
for (var i = 0; i < figures.length; i++) {
    figures[i].onclick = (e) => {
        if (e.target.nodeName === "IMG") {
            viewer.show(e.target.src);
        }
    };
}
var containers = document.querySelectorAll("#iv-container .iv-image-view");
for (var i = 0; i < containers.length; i++) {
    containers[i].onclick = () => {
        if (!dragged) {
            viewer.hide();
        }
    };
}
```

7. Open external links

When following links, to remain the blog page opened, external links should be shown in new tabs without any tracking information. To do that, I write some lines of code to get all external links in the page, then set `target = "_blank"` and add attribute `rel = "noopener noreferrer"` to them.

assets\extra.js

```
/* open external links in new tab */
var links = document.links;
for (var i = 0, linksLength = links.length; i < linksLength; i++) {
    if (links[i].hostname != window.location.hostname) {
        links[i].target = "_blank";
        links[i].setAttribute("rel", "noopener noreferrer");
        links[i].className += " externalLink";
    } else {
        links[i].className += " localLink";
    }
}
```

8. Custom styles

After all extensions and plugins are installed, some extra pages and elements are added, this is the time to tweak the whole site's styles.

8.1. Colors

Here are some small additional styles to make the theme look a bit harmonious with the selected theme color

- Logo and headers should be in orange to be highlighted, and active links can be in dark blue:

```css
.md-logo,
.md-typeset h1 {
    color: orangered;
}
.md-typeset h2,
.md-typeset h3,
.md-typeset h4,
.md-tabs__link.md-tabs__link--active,
.md-nav__link.md-nav__link--active {
    color: darkblue;
}
```

- Search input should have white background color:

```css
.md-search__input {
    background-color: white !important;
}
```

- Non-highlighted code needs stand out a bit in dark red in white background:

```css
.md-typeset code {
    color: darkred;
    background-color: rgba(0, 0, 0, 0.01);
}
```

- Normal paragraph should be fully justified:

```css
.md-typeset p {
    text-align: justify;
}
```

- Emphasized text should be in dark magenta:

```css
.md-typeset em {
    color: darkmagenta;
}
```

- Footer should look smaller by changing the background color:

```css
.md-footer {
    color: unset;
```

```
        background-color: unset;
    }
    .md-footer-meta {
        background-color: black;
    }
```

- The highlight color should not be too yellowish, I'd like to reduce its opacity:

```
:root > * {
    --md-code-hl-color: rgba(255, 255, 0, 0.1);
    --md-typeset-mark-color: var(--md-code-hl-color);
}
```

8.2. Admonition

I want to make admonitions look more harmonious to the theme, so I decided to remove border and shadow box, then add a light background color which is the title background color of each type. The font and the margin also need modified a bit to make admonitions look consistent to the main content.

First, icon should be bigger:

```
.md-typeset .admonition-title:before,
.md-typeset summary:before {
    height: 1.2rem;
    width: 1.2rem;
}
```

Remove border, shadow box, and increase font size:

```
.md-typeset .admonition,
.md-typeset details {
    border: none;
    box-shadow: none;
    font-size: 0.95em;
    margin-top: 0;
}
```

Remove background color in the title, step it back to the right, due to the increased icon size:

```
.md-typeset .admonition > .admonition-title,
.md-typeset details > summary {
    background-color: transparent !important;
    border-left: 0.2rem solid transparent !important;
    padding-left: 3em;
}
```

Fill background color for different admonition types:

```css
.md-typeset .admonition.note,
.md-typeset details.note {
    background-color: rgba(68, 138, 255, 0.05);
}
.md-typeset .admonition.abstract,
.md-typeset .admonition.summary,
.md-typeset .admonition.tldr,
.md-typeset details.abstract,
.md-typeset details.summary,
.md-typeset details.tldr {
    background-color: rgba(0, 176, 255, 0.05);
}
.md-typeset .admonition.info,
.md-typeset .admonition.todo,
.md-typeset details.info,
.md-typeset details.todo {
    background-color: rgba(0, 184, 212, 0.05);
}
.md-typeset .admonition.hint,
.md-typeset .admonition.important,
.md-typeset .admonition.tip,
.md-typeset details.hint,
.md-typeset details.important,
.md-typeset details.tip {
    background-color: rgba(0, 191, 165, 0.05);
}
.md-typeset .admonition.check,
.md-typeset .admonition.done,
.md-typeset .admonition.success,
.md-typeset details.check,
.md-typeset details.done,
.md-typeset details.success {
    background-color: rgba(0, 200, 83, 0.05);
}
.md-typeset .admonition.faq,
.md-typeset .admonition.help,
.md-typeset .admonition.question,
.md-typeset details.faq,
.md-typeset details.help,
.md-typeset details.question {
    background-color: rgba(100, 221, 23, 0.05);
}
.md-typeset .admonition.attention,
.md-typeset .admonition.caution,
.md-typeset .admonition.warning,
.md-typeset details.attention,
.md-typeset details.caution,
.md-typeset details.warning {
    background-color: rgba(255, 145, 0, 0.05);
}
.md-typeset .admonition.fail,
.md-typeset .admonition.failure,
```

```css
.md-typeset .admonition.missing,
.md-typeset details.fail,
.md-typeset details.failure,
.md-typeset details.missing {
    background-color: rgba(255, 82, 82, 0.05);
}
.md-typeset .admonition.danger,
.md-typeset .admonition.error,
.md-typeset details.danger,
.md-typeset details.error {
    background-color: rgba(255, 23, 68, 0.05);
}
.md-typeset .admonition.bug,
.md-typeset details.bug {
    background-color: rgba(245, 0, 87, 0.05);
}
.md-typeset .admonition.example,
.md-typeset details.example {
    background-color: rgba(124, 77, 255, 0.05);
}
.md-typeset .admonition.cite,
.md-typeset .admonition.quote,
.md-typeset details.cite,
.md-typeset details.quote {
    background-color: hsla(0, 0%, 62%, 0.05);
}
```

Make the content close to the title a bit:

```css
.md-typeset .admonition > .admonition-title + *,
.md-typeset details > summary + * {
    margin-top: 0;
}
```

Change admonition style

And tweak the style to show admonitions which has icon but do not have title. The trick is to add left padding to the first letter in the content paragraph, and move the content up by applying a negative top margin:

```
.md-typeset .admonition.no-title > .admonition-title + p::first-letter,
.md-typeset details.no-title > summary + p::first-letter {
    padding-left: 2.2em;
}
.md-typeset .admonition.no-title > .admonition-title + *,
.md-typeset details.no-title > summary + * {
    margin-top: -2.1em;
}
```

Use these additional styles, with `.no-title` class and an empty title (use `" "` or `"\ "`):

```
!!! info no-title "\ "

    This admonition has an icon as an inline element with the content

!!! info

    Default title

!!! info "New title"

    Content of the admonition is indented

!!! info ""

    There is no title and no icon
```

ⓘ This admonition has an icon as an inline element with the content

ⓘ **Info**

Default title

ⓘ **New title**

Content of the admonition

There is no title and no icon

8.3. Quotes

Quote is used to provide additional data, so I changed its style a bit to not make it confusing with the main text.

```css
.md-typeset blockquote {
    color: unset;
    border-left-width: 2px;
    opacity: 0.7;
}
.md-typeset blockquote :first-child {
    margin-top: 0.25em;
}
.md-typeset blockquote :last-child {
    margin-bottom: 0.25em;
}
```

> Admonition also has `quote` and `cite` type which are used as cited content.

8.4. Code block

It is better to show the line number inline with its line of code:

```yaml
markdown_extensions:
    - meta
    - pymdownx.superfences
    - pymdownx.highlight:
        linenums_style: pymdownx-inline
```

And then increase the font size to get make them easy to read:

```css
.md-typeset code,
.md-typeset kbd {
```

```
        font-size: 1em;
        word-break: keep-all !important;
    }
    .md-typeset pre code {
        white-space: pre-wrap;
        font-size: 0.9em;
    }
```

Then the line number should be dimmed to not distract the main code:

```
.md-typeset .highlight [data-linenos]:before {
    background-color: transparent;
    box-shadow: none;
    color: lightgray;
}
```

```
2    #include <stdio.h>
3
4    int main(void) {
5        printf("Hello world!\n");
6        return 0;
7    }
```

8.5. Tables

Table should show cell border and use a full width to make the content clear.

```
.md-typeset__scrollwrap {
    margin: unset;
}
.md-typeset__table {
    padding: 0;
    display: block;
}
.md-typeset table:not([class]) {
    font-size: 0.9em;
    box-shadow: none;
    display: table;
    border-collapse: collapse;
}
.md-typeset table:not([class]) th,
.md-typeset table:not([class]) td {
    padding: 0.5em;
    border: 1px solid #f0f0f0;
    min-width: unset;
}
```

Syntax	Description	Test Text

Syntax	Description	Test Text
Left align	Center align	Right align
A text	Another text	More texts

8.6. Tabs

Tab's content should show some intent to visualize its structure. The label is tweaked a bit to look consisted with its content at the left margin. Note that in printing, all tabs will be expanded to show all of its content.

```css
.md-typeset .tabbed-content {
    box-shadow: none;
    border-top: 1px solid var(--md-default-fg-color--lightest);
}

.md-typeset .tabbed-content > .tabbed-set {
    padding-left: 2em;
}

.md-typeset .tabbed-set > label {
    font-size: 0.9em;
    padding: 0
    margin-right: 1.25em;
}
```

Tab 1

Some texts

Tab A

Text A

Tab B

Text B

Tab 2

Some other texts

8.7. Buttons

When using white primary color, the default button class `md-button` has issues to display correctly. Here are the fix for those buttons:

- change the margin to make them smaller

- change the border, and text color

- change the size of the icon

```
.md-typeset .md-button {
    font-size: small;
    font-weight: unset;
    padding: 0.25em 0.5em;
    border: 1px solid;
    color: orangered;
}
.md-typeset .md-button .twemoji {
    font-size: large;
}
```

[Click here]

8.8. Image caption

The caption should not have restricted width, and its bottom margin should be smaller.

```
.md-typeset figcaption {
    max-width: unset;
    margin: 1em auto;
}
```


A photo from https://picsum.photos
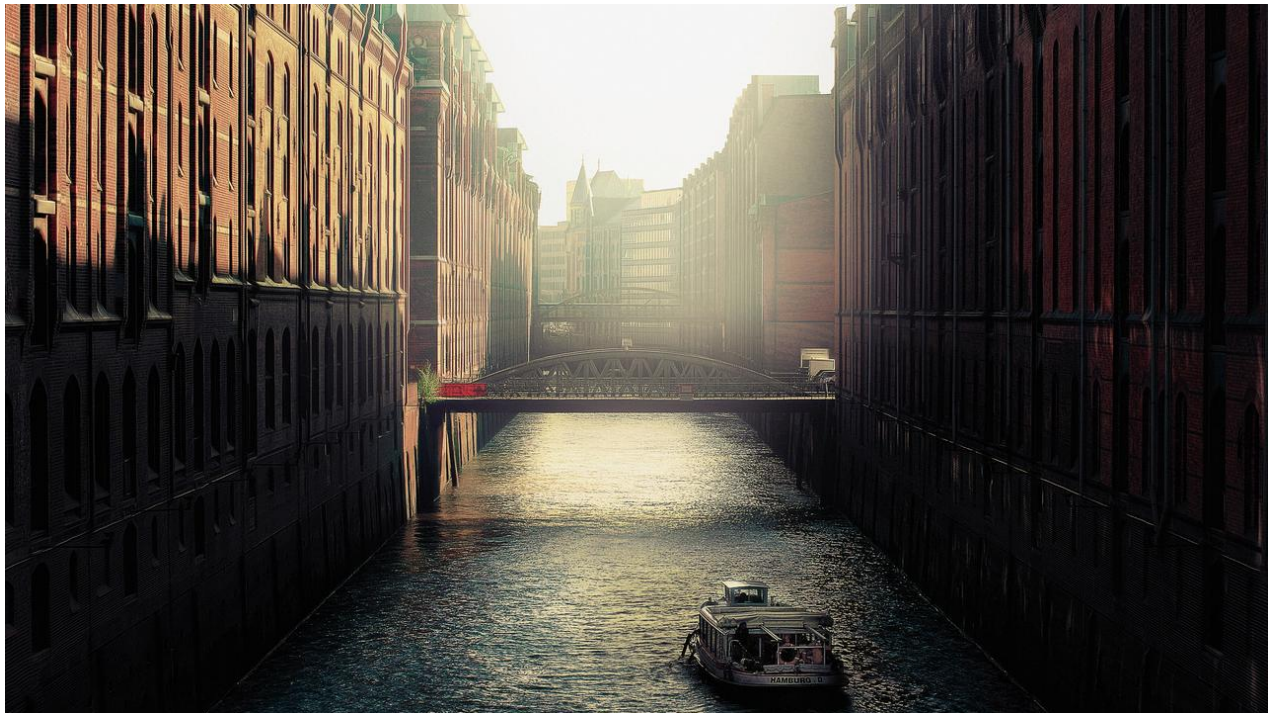
8.9. Sidebar scrolls

Only show the scrollbar when hovering to make sidebar look clear.

```css
.md-sidebar__scrollwrap {
    overflow: hidden;
}
.md-sidebar__scrollwrap:hover {
    overflow-y: auto;
}
```

8.10. Spaces

This part modifies some small space gap and margins to make the overall layout look better.

Remove some space gaps.

```css
.md-main__inner {
    margin-top: 0;
}
md-typeset dd {
    margin: 0em 0 1em 1.875em;
}
.md-typeset dd > * {
    margin-top: 0;
}
.md-typeset p:empty {
    display: none;
}
.md-typeset .admonition + *,
.md-typeset details + * {
    margin-top: 0;
}
/* .md-typeset blockquote, */
.md-typeset dl,
.md-typeset figure,
.md-typeset ol,
.md-typeset pre,
.md-typeset ul {
    margin-top: 0;
    margin-bottom: 0;
}
```

8.11. Tags

Add styles to have a space between tags.

```css
.tag {
    white-space: nowrap;
```

```
    margin-right: 0.25em;
}
```

Then the tag cloud should have some indent:

```
.tag-cloud-content {
    padding: 0 0.6rem;
    margin-bottom: 1em;
}
```

8.12. New elements

8.12.1. New span

Class `.ns` is used in a new span to clear font-style of the target element, when using attribute list on it.

```
.md-typeset .ns {
    font-style: unset;
}
```

An example of adding `.ns` class to an emphasized word:

```
_without new span: italic style_\
_with new span: normal style thanks to `.ns` class_{.ns}
```

without new span: italic style
with new span: normal style thanks to `.ns` class

8.12.2. Row and Column

With Custom Blocks extension, I can make column layout with class `.row` and `.col`. Here are the style to make column layout in a row, and set column's margins:

```
.md-typeset .row {
    display: flex;
    flex-direction: row;
}
.md-typeset .row .col {
    display: flex;
    flex-direction: column;
    width: 100%;
    margin: 0 0.25em;
}
.md-typeset .row .col:first-of-type {
    margin-left: 0;
}
.md-typeset .row .col:last-of-type {
```

```
    margin-right: 0;
}
```

A column will try to fit 100% of the page width. To set a percentage, use the class `.wXX` for the width of `XX%`. These classes can be applied to other elements too.

And here is an example to create 2 columns, a big one is 80% page width:

```
::: row

    ::: col w80 style="background-color: lightyellow;"

        A big column using 80% of page width

    ::: col w20 style="background-color: lightblue;"

        A small column
```

A big column using 80% of page width	A small column