

# Moodular

Use Typst's HTML export to generate content for your Moodle courses

v0.0.1

<https://github.com/SillyFreak/typst-moodular>

**Clemens Koza**

With Typst's upcoming HTML export, a new opportunity for using Typst in teaching opens up: in addition to worksheets and other printed/printable resources, content in learning management systems (LMS) can be created with Typst. This package provides tools for writing materials targeting HTML tailored for Moodle, helps previewing HTML content in the Typst webapp or Tinymist, and integrates the Components for Learning (C4L) that offer a rich visual and semantic language for structuring information.

## CONTENTS

I	Introduction .....	2	II.a moodular .....	4
I.a	HTML preview and export .....	2	II.b c4l .....	6
I.b	Components for Learning (C4L) .....	2		
I.c	Show rules for specific elements .....	3		
I.c.a	Blockquotes .....	3		
I.c.b	raw code blocks .....	3		
I.c.c	Images .....	3		
I.c.d	C4l Blockquotes and figures .....	4		
II	Module reference .....	4		

# I INTRODUCTION

This package allows you to use Typst's HTML export to generate content for your Moodle courses, but preview them faithfully in Tinymist or the webapp, or target PDF in addition to Moodle.

This package is somewhat opinionated, as it contains features tailored towards specific Moodle plugins used at our own school:

- Components for Learning (C4L) (Atto/TinyMCE plugin): Moodular recreates the HTML structure of C4L blocks, and also renders them for PDF export.

## I.a HTML preview and export

one of Moodular's main features is setting up your document for HTML-equivalent preview. This includes using a single unlimited-height page, using a sans serif font for text, and showing links in blue and underlined. Other aspects of this include the display and blockquotes and raw blocks, which are styled to match Moodle's Boost theme's default style.

To get started apply Moodular's settings like this:

```
1 #import "@preview/moodular:0.0.1" as moodular: c4l
2
3 #show: moodular.preview()
4 // or
5 #show: moodular.export()
```

`preview()` is the right choice if you only want HTML export, and use “regular” mode only to preview. If you want to instead target both HTML and PDF exports, `export()` is what you want: it won't change the page setup, but apply other styles. Both are just wrappers around `setup()` the exact behavior is documented there.

## I.b Components for Learning (C4L)

A huge part of this package is providing the Components for Learning (C4L) features. Most but not all components are currently supported:

### Contextual components

- ✓ Do/Don't Cards
- ✓ Example
- ✓ Figure
- ✗ Inline Tag
- ✓ Key concept
- ✓ Quote
- ✓ Reading Context
- ✓ Reminder

- ✗ Tag
- ✓ Tip

### Procedural components

- ✓ Attention
- ✗ Due Date
- ✗ Estimated time
- ✓ Learning Outcomes
- ✓ Procedural Context

### Evaluative components

- ✓ Expected Feedback
- ✗ Grading Value

### Helper components

- ✓ All-purpose card

The gallery has a document previewing some components, and you can click on the implemented components above to go to their docs, including rendered examples. Please open an issue at <https://github.com/SillyFreak/typst-moodular/issues> if you need one of the missing components.

C4L components generated by the C4L Moodle plugins start with `<p class="c4l-spacer"></p>`, which seems to actually have a negative effect on spacing. By default, Moodular does not remove the spacer to remain consistent with the official plugins, but provides the `c4l.remove-spacer()` function in case this affects you. See that function for details.

## I.c Show rules for specific elements

There are some Typst elements that are transformed by Moodular to either ensure they look like Moodle when exporting to PDF, or that they are picked up correctly by Moodle when exporting to HTML (the latter category may shrink as Typst's HTML support matures).

### I.c.a Blockquotes

Blockquotes can e.g. be created by using Markdown syntax in Moodle:

```
1 > quote
```

md

Moodle displays these with lighter colored text and a left gray bar. To make sure a Typst blockquote is displayed the same in both HTML and PDF, Moodular styles them with a show rule when outputting to PDF:

```
1 #quote(block: true)[#lorem(20)]
```

typ

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

### I.c.b raw code blocks

Code blocks can be syntax-highlighted by Moodle if the language is specified. The correct markup for this are nested `<pre class="language-..."><code>` tags. Typst will get these (see [typst#6701](#) and [typst#6702](#)), but for now Moodular transforms raw blocks to the correct shape.

When exporting to PDF, Moodular matches Moodle's background color for code blocks:

```
1 ```py
2 print("Hello, world!") # this is an example
3 ```
```

typ

```
print("Hello, world!") # this is an example
```

### I.c.c Images

Typst's HTML export does not support outputting multiple files, so it can't support images yet. There are two ways in which Moodular can handle this:

#### I.c.c.a frame()

```
1 #moodular.frame(image("../"))
```

typ

Using `frame()`, the contained content is embedded in the output HTML as SVG. Images within that content are base64-encoded. Note that base64 is an inefficient encoding, but for very small images like icons it may be a convenient option.

#### I.c.c.b <img> elements

Images not inside a `frame()` are transformed by Moodular into `<img>` tags during HTML export. Specifically, an image like

```
1 #moodular.frame(image("example.png", alt: "Example"))
```

typ

is transformed into the following:

```
1  html
```

Here, @@PLUGINFILE@@ is a prefix internally used by Moodle to identify attachments referenced inside rich text editors. What you should know about this is:

- When you are first transferring your HTML to Moodle, the image will naturally not be found. You will need to create an `<img>` by uploading the image through the rich text editor's "insert image" option. That should create an element like this:

```
1 
```

- Moodle can now (after uploading the image; saving is not necessary) identify the image using the `example.png` filename. In fact, when saving the resource, the rest of the URL is removed by Moodle and replaced by @@PLUGINFILE@@ because Moodle only needs this part for editing.
- When you replace that image by what Moodle created, the preview will no longer display the image (since the `src` changed from a working draft URL on your Moodle site to a placeholder containing @@PLUGINFILE@@), but this does not impact how the image is saved when saving the whole resource. So, upon saving, you should see your content, including the uploaded and properly referenced image.
  - Note that the width and height attributes of the `<img>` tag have not been preserved; Moodle currently doesn't offer a solution for this, but these missing attributes should mostly affect page layout while an image is still loading and not much more.
- When you now edit the same resource again, all images will be shown (the placeholder has been replaced by a draft URL again). Assuming your new content does not contain previously unknown images, you can simply replace the whole HTML by what Moodle produced again, and all uploaded images will be preserved.

### I.c.d C4L Blockquotes and figures

Apart from Typst's `#quote(block: true)` which is by default styled to match Moodle, C4L also provides a quote component, available as `c4l.quote()`. If you want all quotes in your document shown that way, you can use `c4l.blockquotes-as-c4l()`; see that function for more details.

Likewise, C4L provides a figure component. You can apply `c4l.figures-as-c4l()` to your document to apply that component to all regular figures. Note that C4L figures differ from Typst's only in HTML output; the PDF output is exactly the same.

## II MODULE REFERENCE

### II.a modular

- |   |  |
|---|--|
| <ul style="list-style-type: none"><li>• <code>setup()</code></li><li>• <code>preview()</code></li></ul> | <ul style="list-style-type: none"><li>• <code>export()</code></li><li>• <code>frame()</code></li></ul> |
|---|--|

```
setup(mode: string) -> function
```

Moodular's main template function, to be called as a show rule:

```
1 #show: setup(mode: "preview") // or "export"
```

typ

If you don't need flexibility of the mode, you can use `preview()` or `export()` instead.

The function applies the following settings for PDF export:

- if mode is "preview", sets the page height to auto so that all content appears on a single continuous page, like on the web
- sets a sans serif font (Noto Sans is currently hardcoded and therefore needs to be present) and displays links blue and underlined
- shows blockquotes with a gray left border
- show raw blocks with a light beige background

which is a best-effort recreation of Moodle's Boost theme's default style.

For HTML export, the following settings are applied:

- raw blocks are wrapped in `<pre class="language-..."><code>` tags to apply Moodle's syntax highlighting
- image elements are transformed into `` tags. The `@@PLUGINFILE@@` prefix is used internally by Moodle to indicate references to uploaded files.

#### Parameters:

`mode (string = none)` – the mode to use for the document; either `preview` (simulate a non-paged run of HTML content) or `export` (don't change the page setup, as to produce a paged PDF document).

```
preview() -> function
```

Sets up the document for HTML preview, i.e. simulating a non-paged run of HTML content. Call this as a show rule:

```
1 #show: preview()
```

typ

This is a wrapper around `setup()`, see that function for details.

```
export() -> function
```

Sets up the document for PDF export, i.e. producing a paged PDF document. Call this as a show rule:

```
1 #show: export()
```

typ

This is a wrapper around `setup()`, see that function for details.

```
frame(body: content) -> content
```

Conditionally puts content into an `html.frame`, i.e. rendering it as an `<svg>` tag. When exporting to PDF, the content is simply displayed as-is.

#### Parameters:

`body (content)` – the content to put into a frame

## II.b c4l

- `remove-spacer()`
- `key-concept()`
- `tip()`
- `reminder()`
- `quote()`
- `do-dont()`

- `reading-context()`
- `example()`
- `figure()`
- `attention()`
- `procedural-context()`
- `learning-outcomes()`

- `expected-feedback()`
- `card()`
- `blockquotes-as-c4l()`
- `figures-as-c4l()`

```
remove-spacer(value: bool) -> content
```

Enables or disables the removal of `<p class="c4l-spacer"></p>` elements from the output when exporting to HTML. By default, these elements are rendered to be consistent with the C4L editor moodle plugins (for Atto and TinyMCE), but this looks bad according to my testing on a clean Moodle install; see also [https://github.com/reskit/moodle-tiny\\_c4l/issues/20](https://github.com/reskit/moodle-tiny_c4l/issues/20).

The default of this setting can be overridden by compiling with `--input modular-remove-spacer=true`.

#### Parameters:

`value (bool)` – whether to remove spacer elements or not.

```
key-concept(body: content, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/key-concept/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.key-concept[#lorem(20)]
```

typc

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua quaerat.

#### Parameters:

`body (content)` – the body to wrap in the component

`full-width (bool = false)` – whether the component should take up the full text width

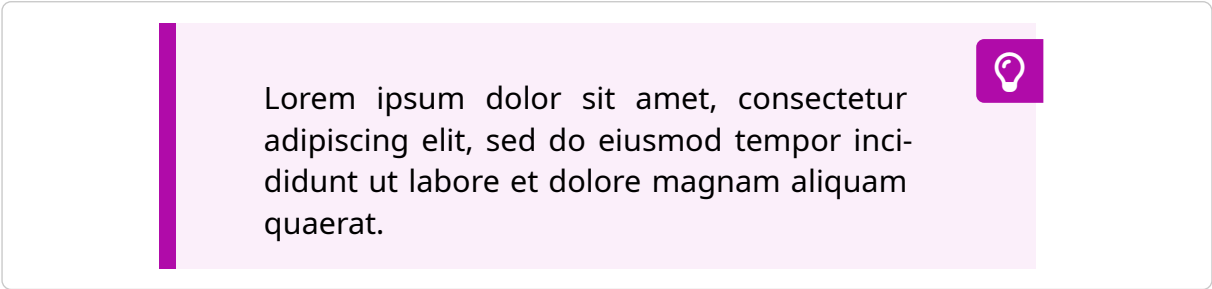
```
tip(body: content, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/tip/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.tip[#lorem(20)]
```

typc

A preview of the 'tip' component. It features a light purple rectangular box with a thin vertical purple bar on the left side. Inside the box, the text 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.' is displayed in a black serif font. In the top right corner of the box, there is a small purple square icon containing a white lightbulb symbol.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

#### Parameters:

`body (content)` – the body to wrap in the component

`full-width (bool = false)` – whether the component should take up the full text width

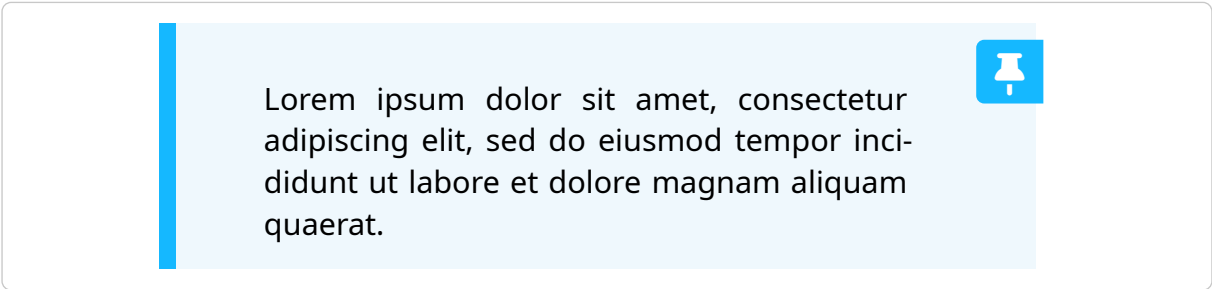
```
reminder(body: content, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/reminder/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.reminder[#lorem(20)]
```

typc

A preview of the 'reminder' component. It features a light blue rectangular box with a thin vertical blue bar on the left side. Inside the box, the text 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.' is displayed in a black serif font. In the top right corner of the box, there is a small blue square icon containing a white pushpin symbol.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

#### Parameters:

`body (content)` – the body to wrap in the component

`full-width (bool = false)` – whether the component should take up the full text width

```
quote(body: content, full-width: bool, attribution: content | none) -> content
```

See <https://componentsforlearning.org/components/quote/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.quote[#lorem(20)]
```

typc

*“Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.”*

```
1 c4l.quote(attribution: lorem(5))[#lorem(20)]
```

typc

*“Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.”*

Lorem ipsum dolor sit amet.

#### Parameters:

body (content) – the body to wrap in the component

full-width (bool = false) – whether the component should take up the full text width

attribution (content or none = none) – the attribution associated with the quote

```
do-dont(do: content, dont: content, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/do-dont-cards/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.do-dont[#lorem(20)][#lorem(20)]
```

typc



Lorem ipsum dolor sit amet, consectetur adipiscing elit,  
 sed do eiusmod tempor incididunt ut labore et dolore  
 magnam aliquam quaerat.



Lorem ipsum dolor sit amet, consectetur adipiscing elit,  
 sed do eiusmod tempor incididunt ut labore et dolore  
 magnam aliquam quaerat.



### Parameters:

`do ( content )` – the body to wrap in the component’s “do” block

`dont ( content )` – the body to wrap in the component’s “don’t” block

`full-width ( bool = false )` – whether the component should take up the full text width

```
reading-context(  
  body: content ,  
  full-width: bool ,  
  attribution: content none ,  
  comfort-reading: bool ,  
) -> content
```

See <https://componentsforlearning.org/components/reading-context/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.reading-context(comfort-reading: true)[#lorem(20)]
```

typc

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do  
 eiusmod tempor incididunt ut labore et dolore magnam aliquam  
 quaerat.

```
1 c4l.reading-context(attribution: lorem(5))[#lorem(20)]
```

typc

Lorem ipsum dolor sit amet, consectetur adipiscing elit,  
sed do eiusmod tempor incididunt ut labore et dolore  
magnam aliquam quaerat.

*Lorem ipsum dolor sit amet.*

#### Parameters:

body (content) – the body to wrap in the component

full-width (bool = false) – whether the component should take up the full text width

attribution (content or none = none) – the attribution associated with the quote

comfort-reading (bool = false) – whether to display the context in a serif font

```
example(title: content, body: content, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/example/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.example[Title][#lorem(20)]
```

typc

#### TITLE

Lorem ipsum dolor sit amet, consectetur adipiscing  
elit, sed do eiusmod tempor incididunt ut labore et  
dolore magnam aliquam quaerat.

**Parameters:**

`title (content)` – the example’s title

`body (content)` – the body to wrap in the component

`full-width (bool = false)` – whether the component should take up the full text width

```
figure(..args: arguments, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/figure/> for details and examples rendered in HTML.

This is rendered as a regular figure in PDF preview:

```
1 c4l.figure(rect())
```

typc



```
1 c4l.figure(rect(), caption: lorem(5))
```

typc



Lorem ipsum dolor sit amet.

**Parameters:**

`..args (arguments)` – the arguments accepted by Typst’s built-in `figure()`

`full-width (bool = false)` – whether the component should take up the full text width

```
attention(body: content, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/attention/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.attention[#lorem(20)]
```

typc

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.



### Parameters:

body (content) – the body to wrap in the component

full-width (bool = false) – whether the component should take up the full text width

```
procedural-context(body: content , full-width: bool ) -> content
```

See <https://componentsforlearning.org/components/procedural-context/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.procedural-context[#lorem(20)]
```

typc

*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.*

### Parameters:

body (content) – the body to wrap in the component

full-width (bool = false) – whether the component should take up the full text width

```
learning-outcomes(title: content , body: content , full-width: bool ) -> content
```

See <https://componentsforlearning.org/components/learning-outcomes/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.learning-outcomes[Title][
2   - #lorem(5)
3   - #lorem(10)
4 ]
```

typc

**TITLE**

- ▶ Lorem ipsum dolor sit amet.
- ▶ Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

```
1 c4l.learning-outcomes[Title][
2   + #lorem(5)
3   + #lorem(10)
```

typc

4 ]

TITLE

1. Lorem ipsum dolor sit amet.
2. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

#### Parameters:

`title (content)` – the block's title

`body (content)` – the body to wrap in the component

`full-width (bool = false)` – whether the component should take up the full text width

```
expected-feedback(body: content , full-width: bool ) -> content
```

See <https://componentsforlearning.org/components/expected-feedback/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.expected-feedback[#Lorem(20)]
```

typc

*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua quaerat.*



#### Parameters:

`body (content)` – the body to wrap in the component

`full-width (bool = false)` – whether the component should take up the full text width

```
card(body: content, full-width: bool) -> content
```

See <https://componentsforlearning.org/components/all-purpose-card/> for details and examples rendered in HTML.

This is how the component looks in PDF preview:

```
1 c4l.card[#lorem(20)]
```

typc

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua quaerat.

#### Parameters:

body (content) – the body to wrap in the component

full-width (bool = false) – whether the component should take up the full text width

```
blockquotes-as-c4l(full-width: bool) -> function
```

An optional show rule that transforms all regular Typst blockquotes into C4L quotes:

```
1 #quote(block: true)[before ...]  
2 #show: c4l.blockquotes-as-c4l()  
3 #quote(block: true)[... after]
```

typ

before ...

“... after”

#### Parameters:

full-width (bool = false) – whether blockquotes should take up the full text width

```
figures-as-c4l(full-width: bool) -> function
```

An optional show rule that transforms all regular Typst figures into C4L figures:

```
1 #figure(image("example.png"), caption: [before ...])  
2 #show: c4l.figures-as-c4l()  
3 #figure(image("example.png"), caption: [... after])
```

typ

Note that C4L figures are displayed as regular figures when exporting to PDF, so there is no difference to show here. The generated HTML would look roughly like this:

```
1 <figure> html
2   
3   <figcaption>Figure 1: before ...</figcaption>
4 </figure>
5 <p class="c4l-spacer"></p>
6 <figure class="c4lv-figure" aria-label="Figure">
7   
8   <figcaption><em class="c4l-figure-footer">Figure 2: ... after</em></figcaption>
9 </figure>
```

### Parameters:

full-width (bool = false) – whether figures should take up the full text width