Kian's Homework Template

https://github.com/kdkasad/typst-homework-template

Kian Kasad

2025-01-15

Contents

Examples

1 Introduction

This project is a document template I created for homework assignments. It is licensed under the BSD-3-Clause license and is available at the URL on the cover page.

2 Using the template

To use the template, download the file `khw.typ` from this project's repository and place it in your project/document's working directory. If your project is a Git repository, you can use a Git submodule to include the template repository within your own.

2.1 Document setup

To set up your document, import the `khw()` function and apply it to all content using a show rule:

Markup

```
#import "khw.typ": khw
#show: khw.with(
  title: [My Homework Assignment],
  author: "Your Name",
)
```

Example 1: Document setup code.

The `title` and `author` fields are used to print a title block on the first page, set the document's metadata using `document()`, and print a header on every page except the first.

The `khw()` function supports the following optional parameters:

`title` (`content` | `str` | `none`) Document title. Defaults to `none`. If `none`, no title block is printed and no document metadata is set.

`author` (`content` | `str` | `none`) Document author. Defaults to `none`.

`date` (`datetime`) Document creation date. Defaults to `datetime.today()`.

`course` (`content` | `string` | `none`) Course for which the document is being written. Defaults to `none`. See also Section 2.2 for details on how the course name is shortened for use in the page header.

The following options do not take effect immediately, but are used to set default options for the `problem()` and `parts()` functions provided by this template.

`newpages` (`bool`) Whether to start each problem on a new page. Defaults to `false`.

`problem-prefix` (`str`) The word to place before each problem number. Defaults to "Problem".

`align-numbers` (`alignment`) How to align problem numbers within the problem header. Default is `center + horizon`.

`parts-numbering` (`str` | `function`) Numbering to use for parts of a problem. Takes a value which can be used as the argument to `numbering()`. Defaults to `"(a)"`, resulting in parts numbered (a), (b), (c), etc.

Note that passing a function here will not work (currently). This appears to be because Typst's `state` type cannot handle storing functions, but I have not verified this.

2.2 Page header

On every page except the first, a header will be displayed, consisting of the course and title on the left and the author on the right.

The course is preprocessed in a special way: if the `course` parameter to the `khw()` function was a string, a *course prefix separator* and everything after it is removed. The course prefix separator is anything that matches the following regular expression: `(:\s+|\s+-+\s+)`. This allows you to print something like "CS 250: Computer Architecture" in the title block while still keeping the header short enough. In this case, only "CS 250" would appear in the page header.

2.3 Typesetting problems

Use the `problem()` function to typeset a problem. The function takes a non-optional content argument which can be used to specify the problem prompt/question.

Markup

```
#import "khw.typ": problem
#problem[
  #lorem(25)
]
```

Result

Problem 1

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

Example 2: Typesetting a problem using `problem()`.

If the `outlined` parameter is not disabled, a first-level heading is created for each problem. If you need headings within problems, you should start with second-level headings. This also means that problems will appear in the table of contents (if your document has one) and in the table of contents embedded in the PDF's metadata (a.k.a. bookmarks).

The `problem()` function takes the following optional arguments:

`prefix` (`auto` | `str` | `content` | `none`) Specifies the word to place before the problem number. When `auto`, the value of the `problem-prefix` argument to the `khw()` function is used. Defaults to `auto`.

Markup

```
#problem(prefix: "Exercise")[
  #lorem(25)
]
```

Result

Exercise 2

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

Example 3: Problem with prefix specified.

`number` (`auto` | `str` | `content`) Specifies the number of the problem. When `auto`, problems are automatically numbered sequentially starting from 1. Defaults to `auto`.

`points` (`none` | `str` | `int` | `float` | `content`) Specifies the point value of the problem. Defaults to `none`. See Example 4.

Markup

```
#problem(points: 5)[
  #lorem(25)
]
```

Result

Problem 3

(5 points) Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus.

Example 4: Problem with points value specified.

`newpage` (`auto` | `bool`) Whether to insert a page break before the problem header. If `auto`, the value of the `newpages` argument to the `khw()` function is used. Defaults to `auto`.

`align-number` (`auto` | `alignment`) How to align the problem number. When `auto`, the value of the `align-numbers` argument to the `khw()` function is used.

Passing a value of `center + top` will center the problem number right below the problem prefix text, so that it always appears consistent no matter the height of the problem prompts.

`outlined` (`bool`) Whether this problem shows up in the outline. If `true`, an invisible `heading()` is created for this problem, making it act like a regular heading. Defaults to `true`.

2.4 Typesetting multi-part problems

For problems with multiple parts, there are two places you might want to typeset the parts: in the prompt and in the response/solution.

2.4.1 In the prompt

The content which makes up the prompt is displayed with a `set` rule that numbers regular lists using the format `"(a)"`. You can create a regular numbered list in the prompt to typeset a multi-part prompt. See Example 5.

Markup

```
#problem(points: 10)[
  Explain how virtual memory speeds up the following operations:
  + Allocating zero-initialized pages.
  + Spawning child processes.
]
```

Result

Problem 4

(10 points) Explain how virtual memory speeds up the following operations:

(a) Allocating zero-initialized pages.

(b) Spawning child processes.

Example 5: Multi-part problem prompt.

2.4.2 In the response

Since the solution is more likely to include regular numbered lists, I decided not to just use a `set` rule and to instead make a function for typesetting parts. This also makes it quite easy to split up multi-part problems into separate source files.

Use the `parts()` function to typeset multiple parts in the solution. See Example 6.

Markup	`#parts[`
	`  Virtual memory allows for copy-on-write behavior, making`
	`  page allocation faster.`
	`][`
	`  Only the page table needs to be copied when spawning a child`
	`  process, rather than the entire space of mapped memory.`
	`]`

Result	(a) Virtual memory allows for copy-on-write behavior, making page allocation faster. (b) Only the page table needs to be copied when spawning a child process, rather than the entire space of mapped memory.

Example 6: Multi-part problem response.

Sub-parts can be typeset using a normal numbered list and will be numbered with lowercase Roman numerals. See Example 7.

Markup	`#parts[`
	`  #lorem(5)`
	`  + #lorem(5)`
	`  + #lorem(5)`
	`]`

Result	(a) Lorem ipsum dolor sit amet. i. Lorem ipsum dolor sit amet. ii. Lorem ipsum dolor sit amet.

Example 7: Solution with parts and sub-parts.

2.5 To-do markers

You can use the `todo` variable to mark things you need to come back to later.

Markup	`#import "khw.typ": todo`
	
	`Here is some text. #todo`

Result	Here is some text. [To do]

Example 8: The to-do marker.

3 Customization

In order to make more complicated customizations without having to modify `khw.typ`, many of the elements produced by the document template are labeled. You can use `show` rules targeting the labeled elements to change their appearance or override them completely.

Markup

```
#show <khw-problem-points>: smallcaps
#show <khw-problem-prompt>: set text(blue)
#show <khw-problem-number>: it => "#" + it
#problem(points: 2)[
  #lorem(25)
]
```

Result

Problem #5 *(2 POINTS)* Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus.

Example 9: Customizing the template using labels and `show` rules.

3.1 List of labels

`<khw-problem-block>` Attached to the entire block printed at the start of the problem.

`<khw-problem-prefix>` Attached to the problem prefix text.

`<khw-problem-number>` Attached to the problem number. Note that this labels the text itself, not the grid cell which contains the number (and controls its alignment). Unfortunatelly, there is not yet a way in Typst to label the cell itself.

`<khw-problem-points>` Attached to the text which displays a problem's point value.

`<khw-problem-prompt>` Attached to the paragraph which contains the problem prompt.

`<khw-part>` Attached to each element of a `parts()` list. This labels the `block()` containing the part's content.