Lacy-UBC Math Group Project Template User Manual 0.2.0

This Typst template initially is to help you write and format UBC_V MATH 100&101 group projects. It is based on their existing (2025) LaTeX template. Despite the name, it offers a flexible layout for possibly other types of question-solution documents.

This manual has two specifics: one for "**Student:**" who consume documents already populated with content, and the other for "**Instructor:**" (TA's too) who use the template to make projects for the students.

You can skip to the next page if you already know why you are using a Typst template.

The two popular choices among students for typesetting math content, Word and LaTeX, each has significant drawbacks.

Word, sounds familiar and easy, but…
- the math formatting (MathType) may be uncomfortable;
- you pretty much have to use MS Word/WPS/Google Doc/Pages, the editing experience using "libre" solutions like LibreOffice are subpar;
- many do not really know how to make use of Word, like styles, ruler, tab stops and template, so documents get messy.

LaTeX is a powerful and professional academic typesetting system, plus, you can use your favorite text editor, but…
- the syntax may be cryptic;
- the error messages are cryptic;
- compilation (PDF generation) can be slow;

LaTeX	`\[ e^{-\frac{x^2}{3}} \]`	$e^{-\frac{x^2}{3}}$
Typst	`$ e^(-x^2 / 3) $`	

Still in development, Typst is more than enough for our use cases:
- simple math typesetting, much like on WebWork;
- no more `\begin{hell} \backslash \end{hell}`;
- fast compilation, typically in milliseconds;
- friendly manual, function signature and error messages;
- yes, a *fully functional* language server: completion, preview and many more;
- integration with your favorite text editor;
- a modern, free and collaborative online editor
- easy customization by user, relative to LaTeX.

Contents

Getting Started

You have two options: working online or local. Since this is a "group project" template, you probably want to work online for collaboration. Here is a step-by-step guide to get you started.

1. Sign up for an account of the Typst web app.
2. Follow guides and explore a bit.
3. (Optional) Assemble a team.
    1. Dashboard → (top left) Team → New Team.
    2. Team dashboard → (next to big team name) manage team → Add member.

Voilà! You are ready to start your math group project.

Instructor: Initializing a Group Project

To start a math group project in the web app, simply

1. go to the project dashboard;
2. next to "Empty document", click on "Start from a template";
3. search and select "lacy-ubc-math-project";

4. enter your own project name, create, that easy!

In the project just initialized, you will see two files: `config.typ` and `project-1.typ`.

You will likely focus on editing `project-1.typ` for actual questions. The `config.typ` file can contain group and theme information that are likely more useful to students. You may also edit and distribute that for reusable theme configuration.

`project-1.typ`

A basic start of project looks like

```
#import "@preview/lacy-ubc-math-project:0.2.0": *
#import "config.typ": * // Import the config.
#show: setup.with( //...
```

When you create more project files like `project-2.typ`, `project-3.typ`, copy these topmost two `import`'s and `show`. Below this `#show: setup.with( /*...*/ )` is your project content.

Student: Using a Group Project

Given a project file, populated with typed questions, you may simply insert `solution()` where fit, and start solving. Do not forget to surround elements of a solution with (), if there are more than one; and insert commas between elements.

```
#qns(
  question(
    [What is $1 + 1$?], // ← append a comma, if absent
    solution[It is 2.] // Make sure it is inside the question you are answering
to.
  ),
  question(
      [Good, now solve $integral_0^pi sin(x) dd(x)$.],
      solution[
        Ah yes, this is a classic application of the Quantum Turnip Theorem,
which tells us that for any integral involving sine, the approach is to first
change variables into invisible ducks.
      ]
  )
)
```

---

1. What is $1 + 1$?

   It is 2.

2. Good, now solve $\int_0^\pi \sin(x)\,\mathrm{d}x$.

   Ah yes, this is a classic application of the Quantum Turnip Theorem, which tells us that for any integral involving sine, the approach is to first change variables into invisible ducks.

If the project is provided with a corresponding `config.typ`, consider if it conflicts with yours, if any.

Learning Typst

Yes, you do have to learn it, but it is simple (for our purpose). Consult the Typst documentation, maybe the Typst Examples Book even if they say "don't rely on it."

There is a collection of frequently asked, miscellaneous techniques which many find extremely helpful. That is, if you read Chinese…I guess web translation also works.

Setup

In `setup()`, we define the project details, including the title, group name and authors.

```
#show: setup.with(
  title: [The Project Title],
  group: [The Group Name],
  // The following are from config.typ, keep reading to find out how.
  jane-doe,
  san-zhang,
  // more authors...
)
```

By default, they are displayed like:

THE PROJECT TITLE

The Group Name

Jane **Doe**	**Zhang**san	Fulan **AlFulani**	**Yamada** Hanako
31415926	27182818	31415926	27182818

These title and authors given to `setup()` are also saved to PDF metadata, which is reflected in the PDF document properties.

Caveat At this point, only one name format, "first last", is in the defaults. Contribution is welcome. But, how could Zhangsan and Yamada Hanako work-around their name display? See Advanced: Custom Name Format.

Reusable Content

Since one group can take on multiple projects, it is wise to save common features like the members' information and the group name for multiple uses.

The `config.typ` file is a place to store such data. After the `#import "config.typ": *`, every variable in the file will be visible to you. Looking into the template's `config.typ`, it has

```
#let jane-doe = author("Jane", "Doe", 31415926)
```

which is why we could simply type `jane-doe` in the previous example and pass the full author information.

Student: Author

The `author` function produces an author object, like above. Give it your first name, given name, as the first argument (`"Jane"`), and your last name, family name, surname, as the second argument (`"Doe"`); finally, your student number as the third argument (`31415926`).

In MATH 100/101 group projects we will suffix "NP" to a student's name if they are not present. Assume Jane Doe is absent for the project, simply put

```
#show: setup.with(
  // ...
  jane-doe[NP]
)
```

Math

Formatting math equations is probably the reason you are here.

`$E = m c^2$`	$E = mc^2$
`$e^(i pi) = -1$`	$e^{i\pi} = -1$
`$(-b plus.minus sqrt(b^2 - 4a c)) / (2a)$`	$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

A space is required to display consecutive math letters, like `$m c^2$` for mc^2.

This package has you covered on some common multi-letter operators:

```
Inline math:
$lim_(x->oo), limm_(x->oo)$
$sum_(i=1)^n, summ_(i=1)^n$

Block/display math:
$
  lim_(x -> oo) \
  sum_(i = 0)^n \
  dd(t), dv(,x), dv(y,x)
$
```

Inline math: $\lim_{x\to\infty}$, $\operatorname{lim}_{x\to\infty}$ $\sum_{i=1}^{n}$, $\operatorname{sum}_{i=1}^{n}$

Block/display math:

$$\lim_{x\to\infty} \sum_{i=0}^{n} \mathrm{d}t, \frac{\mathrm{d}}{\mathrm{d}x}, \frac{\mathrm{d}y}{\mathrm{d}x}$$

Caution Though you can, and sometimes want to use block style in inline math, be aware that bigger math expressions occupy more vertical space, separate or overlap with surrounding texts.

For "block" or "display" math, leave a space or newline between *both* dollar signs and the equations.

```
$ E = m c^2 $
```

$E = mc^2$

To break a line in math, use a backslash "\". To align expressions in display math, place an "&" on each line where you want them to align to; you may even use multiple "&"s to align equations that are too long to stay in one part.

```
$
  x + y &= z \
  &= v \
  // place '&'
  // anywhere appropriate
  x = w - y&
$
```

$$x + y = z$$
$$= v$$
$$x = w - y$$

Documented are the built-in math functions and symbols

Texts In Math

To display normal text in math mode, surround the text with double quotes function.

```
$x = "We are going to find out!"$
```

$x = \text{We are going to find out!}$

If you are to display units, see Units and Quantities.

For non-unit, single-character normal text, use `upright()`.

```
$U upright(W) U$
```

UWU

There are other text styles available in math mode.

```
$serif("Serif") \
sans("Sans-serif") \
frak("Fraktur") \
mono("Monospace") \
bb("Blackboard bold") \
cal("Calligraphic")$
```

Serif
Sans-serif
$\mathfrak{Fraktur}$
$\texttt{Monospace}$
$\mathbb{Blackboard\ bold}$
$\mathcal{Calligraphic}$

Numbering and Referencing Equations

Note that you must enable equation numbering to reference equations, which this template does. Attach a `#<label>` right after the equation you wish to reference.

```
$
  e^(i pi) = -1 #<eq:euler>
$
@eq:euler is Euler's identity. \
#link(<eq:euler>)[The same
reference],
```

$$e^{i\pi} = -1 \tag{1.1}$$

Equation 1.1 is Euler's identity.
The same reference,

Extra Math Symbols and Functions

The `physica` package provides additional math symbols and functions.

```
$A^T, curl vb(E) = - pdv(vb(B), t)$
```
$A^T, curl vb(E) = - pdv(vb(B), t)$

```
$tensor(Lambda,+mu,-nu) = dmat(1,RR)$
```
$tensor(Lambda,+mu,-nu) = dmat(1,RR)$

```
$f(x,y) dd(x,y)$
```
$f(x,y) dd(x,y)$

It is imported by this template.

Units and Quantities

Although no as common as in physics, we do sometimes need to use units and quantities. Directly typing the 'units' will not result in correct output.

```
$1 m = 100 c m$
```
$1 m = 100 c m$

```
$N = k g m s^(-2)$
```
$N = k g m s^(-2)$

This template uses the `unify` package for this purpose. If you prefer, you can also import and use the `metro` package.

```
$qty("1", "m") = qty("100", "cm")$
```
$qty("1", "m") = qty("100", "cm")$

```
$unit("N") = unit("kg m s^(-2)")$
```
$unit("N") = unit("kg m s^(-2)")$

As you see, the `qty()` and `unit()` functions correct the numbers, units and spacing.

Caution `unify` does not support content as arguments, so your math content should be made `str` before passing to the quantity and unit functions. The following will not work:

```
$qty(3, Ohm)$
```

Instead, wrap both the number and the unit in double quotes to make them `str`:

```
$qty("3", "Ohm")$
```
$qty("3", "Ohm")$

Question

The `question` function is to create a question object.

```
question(
  [The question.],
  question(
    point: 1,
    label: "special",
    [Sub-question.]
  ),
  question(
    [Another sub-question.],
    question(
      point: 1,
      [Sub-sub-question.],
    ),
    question(
      point: 2,
      [Another sub-sub-question.],
    )
  )
),
```

1. (4 points) The question.
    a. (1 point) Sub-question.
    b. (3 points) Another sub-question.
        i. (1 point) Sub-sub-question.
        ii. (2 points) Another sub-sub-question.

The “4 points” and “3 points” question above had no point specified: parent questions get the sum of their children questions’ points, if their own `point` is left blank.

Referencing Questions

The recommended way is as in the example above, provide a `label` argument, and then you can refer to it using

```
@qs:special
```

Question 1.a.

If provided with a `str`, the label created will automatically have a head, “qs:”, for clarity. Otherwise, nothing is added and you will get what you put in.

For alternative reference text, use the `ref` syntax sugar

```
@qs:special[This special one.]
```

This special one.

or the full syntax

```
#ref(
  <qs:special>,
  supplement: [This special one.]
)
```

This special one.

When a `label` is not provided, this template does attach one for you, based the question number. However, such numbers can easily vary, set a special label for stability.

```
@qs:1-b-ii
```

Question 1.b.ii.

Solution

The `solution` function is to create a solution object.

```
question(
  [Shall I pass MATH 100?],
  solution(
    [You shall #strike[not] pass!],
    // Change target and supplement
    solution(
      target: <qs:special>,
      label: "pass",
      supplement: [*Response to a distant question:*\ ],
      [You can target any question, or even `none`. How cool!]
    )
  ),
)
```

---

1. Shall I pass MATH 100?

   You shall ~~not~~ pass!

   Question 1.a.

   Response to a distant question:
   You can target any question, or even `none`. How cool!

A solution is does not need to be in a question.

Referencing Solutions

Similar to the questions, you may provide solutions with unique label in order to reference them. They are not automatically labelled.

`@sn:pass \` `@sn:pass[Pass.]`	Solution to Question 1.a. Pass.

Note that, when a solution does have a target question, the default reference text contain a link to the question as well. Hence, if you click on the part after "to…", it jumps to the question instead of the solution.

However, those with custom supplement refer to only the solution.

Config

There is a default config, `defaults`, and a `config` that you import from `config.typ`, suppose you followed the template.

For the configs you give, the first to the last and one by one, the latter's entries replaces the former's in case of duplication. The "zero-th" config is `defaults`.

A set of config is called a theme. The package has a `theme` module, which of course contains built-in themes; PR is welcome!

To apply a theme, simply put

```
setup.with(
  // ...
  config: theme.ubc-light, // Obviously, a UBC package comes with UBC themes.
)
```

To merge your own config with the theme, and let your config take priority,

```
setup.with(
  // ...
  config: (
    theme.ubc-light,
    config, // The latter's entries replace the formers!
  )
)
```

Besides, you can pass configs to individual components of this package, such as `author`, `question` and `solution`. They need the full config, not just their respective entry. In addition, components in the `qns` wrapper will pass their config down to children components.

Built-in Themes

Besides the default theme, there are:

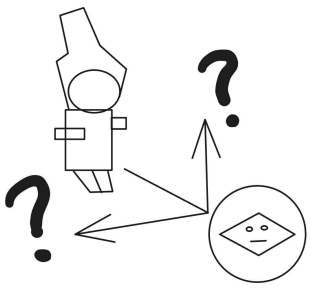
`ubc-light` A small dose of UBC theme colors based on UBC Brand Identity Rules

`ubc-dark` The same idea, but colors are switched for dark mode.

Drawing

Typically, you would not want to commit time and effort to learn drawing in Typst. Have your graphs done in Desmos, GeoGebra, or whatever, then display images of them.

```
#image("template/assets/madeline-math.jpg", width: 6cm)
```



Note that Typst cannot reach beyond the project root directory, so put your assets inside the project folder.

Instructor: Drawing in Typst

Typst has native drawing capability, but quite limited. There is an ad hoc Typst drawing library, a package actually, called “cetz”.

In this template,

```
#import drawing: *
```

to make `cetz` and other drawing helpers available.

For data visualization, use lilaq. For generic drawing, use cetz. For generic plotting, use cetz-plot.

There are other drawing packages available, but not included in the `drawing` module, here is a brief list:

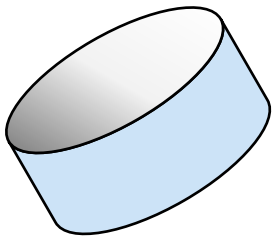
- fletcher: nodes & arrows;
- jlyfish: Julia integration;
- neoplot: Gnuplot integration.

Find more visualization packages here.

Instructor: Template Helpers

The `drawing` module has its own drawing helpers. For example, the `cylinder()` function draws an upright no-perspective cylinder:

```
#import drawing: *
#cetz.canvas({
  import cetz.draw: *
  group({
    rotate(30deg)
    cylinder(
      (0, 0), // Center
      (1.618, .6), // Ellipse radius
      2 / 1.618, // Height
      fill-top: gradient.linear(
        black.lighten(50%),
        black.lighten(95%),
        white,
        angle: -60deg
      ), // Top color
      fill-side:
blue.transparentize(80%), // Side
color
    )
  })
})
```



Example

Below is a more elaborate drawing.

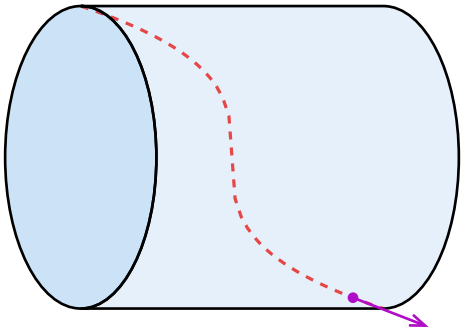


Figure 1: Adaptive path-position-velocity graph

Advanced

This part assumes familiarity of Typst.

Feeder

There is a special component, `feeder`. Like `question` and `solution`, it is used in `qns` and can contain children components.

However, its also takes a positional argument `proc`. `proc` is an arbitrary function, the only requirement is that it should be able to take in all its components and named arguments: the components will first be "visualized", meaning converted to visual content, then passed to `proc` positionally; the named arguments of the `feeder` are passed as named arguments.

```
feeder(
  table.with(
    columns: 2,
  ),
  stroke: blue,
  question[What can I know?],
  question[What should I do?],
  [What may I hope?]
)
```

1. What can I know?	2. What should I do?
What may I hope?	

Internals

The `internal` module allows access to all internal variables—fields and functions that are not of the ordinary user interface.

```
#repr(dictionary(internal))
```

```
(
  spec: <module spec>,
  util: <module util>,
  components: <module
components>,
  defaults: <module defaults>,
  drawing: <module drawing>,
  markscheme: <module
markscheme>,
)
```

You can call all the internal functions, just read the friendly manual, a.k.a. the function signatures, or you would mess up.

```
#internal.components.target-
visualizer(
  <qs:special>,
  t => ref(t, supplement: repr(t))
)
```

<qs:special>

Config

`defaults` Namespace

We already have the `defaults` dictionary, which looks like: `(`

```
  link: (color-major: rgb("#005198"), rule: (..) => ..),
  ref: (color-major: rgb("#1c7a26"), rule: (..) => ..),
)
```

The `internal.default` module is not the same, because the apparent `defaults` is just the `config` field of the `defaults` module, taking over the name.

Propagating Config

Since only the `config` field is taken when you provide a module as config, the process to make `config` or other variables in the module does not matter. You are free to do whatever in your config file (then imported as a module) to propagate that config.

Author

Special Name Content

In some (unlikely) cases, one's name cannot be converted to plain text. Take Ga***lli***$\overline{leo}$ as an example. The name is so special that it cannot be converted to plain text. You may provide a `plain` version of it to avoid incomprehensible PDF metadata.

```
author(
  [#underline(text(fill: purple)[Ga])#strike[*_lli_*]#overline($cal("leo")$)],
  "Smith",
  12345678,
  plain: "Gallileo Smith"
)
```

Custom Name Format

The `author` function accepts a `config`, in which the format of name, affixes, containers and show rules are defined. Namely, `name-format`, `affix-format`, `container`, `set-container` and `rule`.

We are particularly interested in `name-format`, because not all names follow the English style.

```
  author(
    "San",
    "Zhang",
    27182818,
    config: (author: (name-format: (f, l, ..s) => [*#l*#lower(f)])),
  ),
```

The `name-format` function is called with a first name and a last name (for now, that is why we leave an `..s` to sink potentially more parameters for compatibility), and is supposed to produce the formatted name.

By default it is `[#first *#last*]`, above we change it to a more conventional Chinese name format, and so it looks.