Lacy UBC Math Group Project Template
User Manual

Contents

Introduction

This template is designed to help you write and format your math group projects. It is based on the existing (2025) LaTex template. Despite the limited initial purpose, it offers a clean layout for possibly other types of question-solution documents.

You are recommended to read the Getting Started and Math.

Motivation

Why use this template? The previous two popular choices, MS Word and LaTex, each had significant drawbacks. MS Word might be easy to start with, but math formatting is a nightmare; plus, software support is not so good on non-Windows platforms. LaTex, on the other hand, is powerful but has a steep learning curve, the source document becomes hardly readable after a few edits; collaboration is not simple as Word, since a free Overlead account can only have one collaborator per document, while an upgrade is (in my opinion) drastically overpriced.

How about elegant math typesetting, blazingly fast automatic layout and unlimited collaboration? Typst seems to be the solution. Although still in development, it is more than enough for our use cases. Let's see:

LaTex	`\[ e^{-\frac{x^2}{3}} \]`	$e^{-\frac{x^2}{3}}$	(1)
Typst	`$ e^(-x^2 / 3) $`	$e^{-\frac{x^2}{3}}$	(2)

Clearly, Typst's math syntax is way more intuitive and readable.

As another plus, Typst documents usually compile in milliseconds, whereas LaTex can take seconds or even longer. With this speed, every keystroke is immediately reflected in the preview, which can be a huge productivity boost.

The official Typst web app allows unlimited collaborators in a document, which is a huge advantage over Overleaf, given that there are often more than 2 people in a math group. Did I mention that team management is also a free feature?

Getting Started

"So, how do I even start using Typst?"

First thing first, it is all free.

You have 2 options: working online or offline. 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 on the Typst web app.
2. Follow some 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.

Initialize Projects

To start a math group project, simply import this package (you should have done it already) and use the `setup()` function and edit `common.typ` to define the project.

Fortunately, you don't have to remember all the details. Typst web app can handle the initialization for you.

In the project dashboard, next to "Empty document", click on "Start from a template", search and select "lacy-ubc-math-project", enter your own project name, create, that easy!

In the project just initialized, you will see 2 files: `common.typ` and `project1.typ`.

If you are to add more projects for the same group, create no new project, but add files to the existing one, like `project2.typ`, `project3.typ`, etc.

common.typ

This file is for common content that can be shared across all projects. For instance, your group name and members.

```
#import "@preview/lacy-ubc-math-project:0.1.1": author
// Modify as you please.
#let authors = (
  jane-doe: author("Jane", "Doe", "12345678"),
  alex-conquitlam: author(
    "Alex",
    "k\u{02b7}ik\u{02b7}\u{0259}\u{019b}\u{0313}",
    99999999,
    strname: "Alex Coquitlam"
  ),
)
#let group-name = [A Cool Group]
// Additional common content that you may add.
#let some-other-field = [Some other value]
#let some-function(some-arg) = { some-manipulation; some-output }
```

project1.typ

Here is where you write your project content.

```
#import "@preview/lacy-ubc-math-project:0.1.1": *
#import "common.typ": * // Import the common content.
#show: setup.with(
  number: 1,
  flavor: [A], // Don't want a flavor? Just remove this line
  group: group-name,
  authors.jane-doe,
  // Say, Alex is absent for this project, so we suffix an "(NP)" to their name.
  authors.alex-conquitlam + (suffix: [(NP)]),
  // If you just want all authors, instead write:
  // ..authors.values(),
)
```

When you create more project files like `project2.typ`, `project3.typ`, copy these topmost two `import`'s and `show`. Below this `#show: setup.with(...)` is your project content.

Questions & Solutions

A math group project mostly consists of questions and solutions. You can use the `question()` and `solution()` functions to structure your content.

```
#question(1)[
  What is the answer to the universe,
life, and everything?
  // The solution should be in the
question.
  #solution[
    The answer is 42.
  ]
  // You can nest questions and
solutions.
  #question[2 points, -1 if wrong][
    What do you get when you multiply
six by nine?
    #solution[
      42\.
    ]
  ]
]
```

1. (1 point) What is the answer to the universe, life, and everything?

   Solution: The answer is 42.

   (a) (2 points, -1 if wrong) What do you get when you multiply six by nine?

   Solution: 42.

Learn Typst

Yes, you do have to learn it, but it is simple (for our purpose).

Here is a quick peek at some useful syntaxes:

```
You will sometimes _emphasize important information_ in your questions and
solutions. // 1 linebreak = 1 space.
Or, go a step further to *boldly* state the matter. <ex:bold> // <label-name> to
place a label.
// 1+ blank lines = 1 paragraph break.

Of course, we write math equations like $x^2 + y^2 = z^2 "with text and
quantities, e.g." qty(2, cm)$. Need big math display?
$ \$ "math" \$ = "display style math" $
$
  E = m c^2 \ // " \" = newline
  lim_(x -> 0) f(x) = 0 #<eq:ex:lim> // Use #<label-name> in math.
$
// #link(<label-name>)[displayed text] to reference a label.
// For equation, figure and bibliography, @label-name is also available.
Want to get #link(<ex:bold>)[*_bold_*]? Let's look at @eq:ex:lim.
```

You will sometimes *emphasize important information* in your questions and solutions. Or, go a step further to **boldly** state the matter.

Of course, we write math equations like $x^2 + y^2 = z^2 \text{ with text and quantities, e.g. } 2\,\mathrm{cm}$. Need big math display?

$$\$ \text{ math } \$ = \text{display style math} \tag{3}$$

$$E = mc^2 \tag{4.1}$$
$$\lim_{x \to 0} f(x) = 0 \tag{4.2}$$

Want to get ***bold***? Let’s look at Equation 4.2.

For general techniques, consult the Typst documentation.

For this template, you can find more help from the “Other helps” line at the bottom of each help section.

Setup

In `setup()`, we define the project details, including the project name, number, flavor, group name, and authors. The displayed title will look like

```
project number, flavor
```

for example,

GROUP PROJECT 1, FLAVOUR A

Then it is the authors. Since this is a "group project" template, `group` indicates the group name, which will be displayed between the title and the authors.

Finally, `authors`. Each author should be a dictionary with `name` and `id`. The `name` should be a dictionary with `first` and `last`. The `id` should be the student number. Such a dictionary can be created with function `author()`. So, it will look like

```
// You are Jane Doe with student number 12345678
author("Jane", "Doe", 12345678),
```

More authors, you ask? Just add more `author()`, separated by commas.

Title and authors made in `setup()` are converted to PDF metadata, which can be seen in the PDF document properties.

Author

The `author()` function is to be used as an argument of the `setup()` function, providing an author dictionary. It takes the first name, last name, and student number as arguments. For example,

```
#show: setup.with(
  author("Jane", "Doe", 12345678),
  // ...
)
```

Inside, the `author()` function will return a dictionary:

`author("Jane", "Doe", 12345678)`	`(name: (first: "Jane", last: "Doe"), id: 12345678, strname: none, suffix: none,)`

```
(
  name: (first: "Jane", last: "Doe"),
  id: 12345678,
  strname: none,
  suffix: none,
)
```

And in the PDF metadata there will be a "Jane Doe" in the authors field, student number not included.

Name Suffix

In MATH 100/101 group projects we will add "NP" next to a student's name if they are not present. The `author()` function has a named argument `suffix` for this purpose.

```
author("Jane", "Doe", 12345678, suffix: "(NP)") // She was not there!
```

However, as we are already using `common.typ` to define authors, it would be easier to add a suffix to an author dictionary on-demand.

```
#show: setup.with(
  authors.jane-doe + (suffix: "(NP)"),
)
```

Special Characters in Names

What if your last name is kʷikʷəƛ̓, that happens to type…

```
k\u{02b7}ik\u{02b7}\u{0259}\u{019b}\u{0313}
```

Well, you still call `author()` with the original name. Hypothesize that

- the audience is not familiar with the name;
- the PDF metadata viewer in use does not support the special characters.

In this case, we can

- provide an English translation of the name;
- use the `strname` argument to specify the English version of the name.

```
author(
  "Alex",
  "k\u{02b7}
ik\u{02b7}\u{0259}\u{019b}\u{0313}
(Coquitlam)",
  12345678,
  strname: "Alex Coquitlam"
)
```

```
(
  name: (
    first: "Alex",
    last: "kʷikʷəƛ\u{313}
(Coquitlam)",
  ),
  id: 12345678,
  strname: "Alex Coquitlam",
  suffix: none,
)
```

If `strname` is set, it will be used in the PDF metadata instead of the displayed name.

In some more extreme cases, `strname` would be a necessity, rather than a backup. Take name Ga~~***lli***~~leo as an example. The name is so special that it cannot be converted to plain text. In this case, you must provide a `strname` to avoid incomprehensible PDF metadata.

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

```
(
  name: (
    first: sequence(
      underline(body: styled(child: [Ga], ..)),
      strike(body: strong(body: emph(body: [lli]))),
      overline(
        body: equation(block: false, body: styled(child: [leo], ..)),
      ),
    ),
    last: "Smith",
  ),
  id: 12345678,
  strname: "Gallileo Smith",
  suffix: none,
)
```

Math

Formatting math equations is probably the reason you are here. Unlike LaTex, math in Typst is simple.

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

$E = mc^2$

```
$e^(i pi) = -1$
```

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

```
$cal(l) = (-b plus.minus sqrt(b^2 - 4a
c))
   / (2a)$
```

$\mathcal{l} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

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

Most of the time, you have to leave a space between single letters to show consecutive letters. The template has you covered on some common multi-letter operators, like

```
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: $\lim_{x\to\infty}$, $\lim\limits_{x\to\infty}$ $\sum_{i=1}^{n}$, $\sum\limits_{i=1}^{n}$

Block/display math:

$$\lim_{x\to\infty} \tag{5.1}$$

$$\sum_{i=0}^{n} \tag{5.2}$$

$$\mathrm{d}t, \frac{\mathrm{d}}{\mathrm{d}x}, \frac{\mathrm{d}y}{\mathrm{d}x} \tag{5.3}$$

Caution Though you can, and sometimes want to use block style in inline math, be aware that the block expressions will occupy more vertical space (clear in the example above), separating lines or overlapping with surrounding texts.

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

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

$$E = mc^2 \tag{6}$$

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.

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

$$\begin{aligned} x + y &= z \qquad (7.1) \\ &= v \qquad (7.2) \\ x = w - y& \qquad (7.3) \end{aligned}$$

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 need normal single-letter text, fist see if it is a lone unit. If so, use the `unit()` function.

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

$\mathrm{N} = \mathrm{kg}\,\mathrm{m}\,\mathrm{s}^{-2}$

A unit with a value is called a quantity, `qty()`.

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

$1\,\mathrm{m} = 100\,\mathrm{cm}$

More about these in Units and Quantities.

Otherwise, use `upright()`.

```
$U space upright(U) space U$
```

$U \; \mathrm{U} \; U$

There are other text styles available in math mode.

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

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

Language Syntax in Math

In (at least) MATH 100 group projects, math equations is a part of your English (or whatever) writings. Make sure to use proper grammar and **punctuations**. Yes, you will add periods after finishing equations.

Numbering and Referencing Equations

Note that you must enable equation numbering to reference equations, which is set by this template. Add a `#<label-name>` right after the equation you wish to reference.

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

$$e^{i\pi} = -1 \tag{8}$$

Equation 8 is Euler's identity.
The same reference.

Extra Math Symbols and Functions

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

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

$A^{\mathsf{T}}, \nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}$

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

$\Lambda^{\mu}{}_{\nu} = \begin{pmatrix} 1 & \\ & \mathbb{R} \end{pmatrix}$

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

$f(x, y)\,\mathrm{d}x\,\mathrm{d}y$

It is imported in 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.

```
$1 m = 100 cm$
```

$1m = 100cm$

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

$N = kgms^{-2}$

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

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

$1\,\mathrm{m} = 100\,\mathrm{cm}$

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

$\mathrm{N} = \mathrm{kg}\,\mathrm{m}\,\mathrm{s}^{-2}$

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

One thing, `metro` does not yet support symbols as units, so if you are to use symbols, make them strings.

```
// This will not work
$qty(3, Omega)$
// ↓ wrap the symbol in double quotes
```

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

$3\,\Omega$

Question

The `question()` function is to create a question block.

```
#question(4)[
  The question.
  #question(2)[
    Sub-question.
  ]
  #question(0)[
    Another sub-question.
    #question(1)[
      Sub-sub-question.
    ] <ex:qs:that-one>
    #question(1)[
      Another sub-sub-question.
    ]
  ]
]
#question[2 points, -2 if wrong][
  The risky bonus question.
]
You see #link(<ex:qs:that-one>)[that
question]?
```

2. (4 points) The question.
   (a) (2 points) Sub-question.
   (b) Another sub-question.
      i. (1 point) Sub-sub-question.
      ii. (1 point) Another sub-sub-question.
3. (2 points, −2 if wrong) The risky bonus question.

You see that question?

Referencing Questions

Questions can be referenced by their automatically assigned labels. For example, question 1.b.ii has label `<qs:1-b-ii>` and can be referenced by `#link(<qs:1-b-ii>)[That question]`. Note that it cannot be referenced by `@qs:1-b-ii`.

If, for some reason, questions with the same numbering occurs multiple times, a number indicating order of occurrence will be appended to the label. For example, the first 1.b will be labeled `<qs:1-b>`, and the second occurrence of numbering will have label `<qs:1-b_2>`.

As you are constructing your project, the numbering automatically assigned to a question may change. If you want a static reference, which will be preferable in most cases, you can assign a custom label to the question.

Just as in the example above, adding a `<label-name>` after the question creates a custom label that would not change with order of questions.

Solution

The `solution()` function is to create a solution block.

```
#solution[
  The solution to the question.
  // Change color, remove supplement
  #solution(color: orange, supplement:
none)[
    Sub-solution.
  ]
  // Change supplement
  #solution(supplement: [*My Answer*: ])[
    Another sub-solution.
  ]
],
```

Solution: The solution to the question.

Sub-solution.

My Answer: Another sub-solution.

Solution is usually put in a question block as a response to it.

```
#question(1)[
  What is the answer to the universe, life, and everything?
  #solution[ The answer is 42. ]
]
```

Hiding Solutions

There are 2 ways to hide solutions.

To disable all solutions (all solutions will not show, no matter what), provide `hide-solution` to compile inputs:

```
typst compile filename.typ --input hide-solution=true
```

The value can be any of `true`, `1`, `yes`, `y`.

This flag is also visible in the `unsafe` module as `__solution-disabled`.

To hide arbitrary solutions, use `toggle-solution()` before the solutions you wish to hide. In this case, individual solutions can be forced to show by setting `force: true` in the `solution()` function.

```
#solution[Visible.]
// toggle solutions off
#toggle-solution(false)
#solution[Hidden.]
// force it to show
#solution(force: true)[Forced to be
visible.]
// toggle them back on
#toggle-solution(true)
#solution[Visible again.]
```

Solution: Visible.

Solution: Forced to be visible.

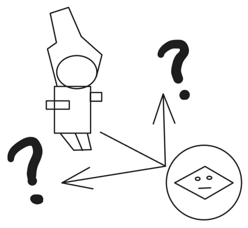
Solution: Visible again.

Drawing

As we are doing math, inevitably we will need to draw some graphs.

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

```
#image("/template/assets/madeline-math.jpg", width: 12em) // n'em = length of n m's
```



You may have noticed that the path in example starts with “/”. It is not your computer’s root directory, but the root directory of the project.

If you are working offline, note that Typst cannot reach beyond the root directory, settable in the compiling command.

Drawing in Typst

So…

Typst has some native drawing abilities, but they are very limited. There is an ad hoc Typst drawing library, a package actually, called “cetz”, with its graphing companion “cetz-plot”. Simply

```
#import drawing: *
```

to let the template import them for you.

For general drawing techniques, refer to the cetz documentation. For graphing, download and refer to the cetz-plot manual.

There are other drawing packages available, but not imported by this template, here is a brief list:

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

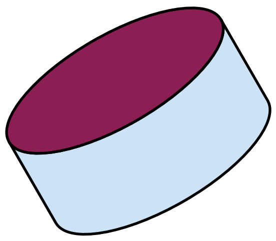
Find more visualization packages here.

Template Helpers

Besides importing the drawing packages, the `drawing` module also provides some helper functions.

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), // Radius: (x, y)
      2cm / 1.618, // Height
      fill-top: maroon.lighten(5%), //
Top color
      fill-side:
blue.transparentize(80%), // Side
color
    )
  })
})
```



Example

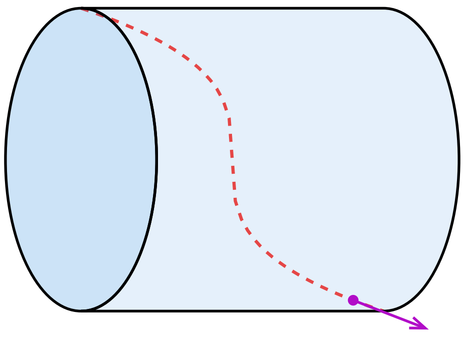


Figure 1: Adaptive path-position-velocity graph (check source code)

Caveats

This package is still in development, so breaking changes might be introduced in the future (you are fine as long as you don't update the compile or the package version).

`unsafe` Module

This template comes with a module called `unsafe`. As obvious, use of its fields or functions is not safe — may break the template. This module is intended for debugging and 'tricks' only. Please make sure that you know what you are doing, should you decide to use it.

Language Support

Though you may use other languages, this template is not optimized for them. In case which the language typesets differently from English, e.g. Chinese and Arabic, you will have to tinker it or accept weird results.

Non-raw Student Number (`id`)

It is possible to use `str` or `content` as a student number for `author()`. This is to be compatible with possible non-numerical formats. When the field can be converted to plain text, it will be displayed as `raw`. Otherwise, the original content will be used, potentially causing inconsistencies. It is recommended that you use `int` or simple `str` for student numbers.

Author Metadata

You are allowed to put `content` as an author's name, as there might be special characters or formatting in the name. However, should anything that Typst cannot convert to plain text be used, the part of the name would not be converted to plain text, and will be replaced by `<unsupported>` when converting to PDF metadata. In that case, you should provide the named argument `strname` to `author()` to specify a plain text version of the name.

Partial Functions

TL;DR: Use the functions like a normal person and as instructed.

Not to be confused with partial application. Some functions in this template are partial, meaning that they are not defined for all possible arguments of the specified type. For example, the `question()` function can break if passed `counters`, `labels` etc. are not right.