Scrutinize

v0.2.0 March 16, 2024

https://github.com/SillyFreak/typst-packages/tree/main/scrutinize

Clemens Koza

Abstract

Scrutinize is a library for building exams, tests, etc. with Typst. It provides utilities for common question types and supports creating grading keys and sample solutions.

Contents

I Introduction

Scrutinize has three general areas of focus:

- It helps with grading information: record the points that can be reached for each question and make them available for creating grading keys.
- It provides a selection of question writing utilities, such as multiple choice or true/false questions.
- It supports the creation of sample solutions by allowing to switch between the normal and "pre-filled" exam.

Right now, providing a styled template is not part of this package's scope.

II Questions and question metadata

Let's start with a really basic example that doesn't really show any of the benefits of this library yet:

```typ
#import "@preview/scrutinize:0.2.0": grading, question, questions

// you usually want to alias this, as you'll need it often
#import question: q

#q(points: 2)[
  == Question

  #lorem(20)
]
```

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

After importing the library's modules and aliasing an important function, we simply get the same output as if we didn't do anything. The one peculiar thing here is `points: 2`: this adds some metadata to the question. Any metadata can be specified, but `points` is special insofar as it is used by the `grading` module. There are two additional pieces of metadata that are automatically available:

- `body`: the complete content that was rendered as the question
- `location`: the location where the question started and the Typst `metadata` element was inserted

The body is rendered as-is, but the location and custom fields are not used unless you explicitly do; let's look at how to do that. Let's say we want to show the points in each question's header:

```typ
#show heading: it => {
  // here, we need to access the current question's metadata
  [#it.body #h(1fr) / #question.current().points]
}
```

Question / 2

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

Here we're using the `question.current()` function to access the metadata of the current question. This function requires context to know where in the document it is called, which a show rule already provides.

III Grading

The next puzzle piece is grading. There are many different possibilities to grade a test; Scrutinize tries not to be tied to specific grading strategies, but it does assume that each question gets assigned points and that the grade results from looking at some kinds of sums of these points. If your test does not fit that schema, you can simply use less of the related features.

The first step in creating a typical grading scheme is determining how many points can be achieved in total, using `grading.total-points()`. We also need to use `question.all()` to get access to the metadata distributed throughout the document:

```typ
#context [
  #let qs = question.all()
  #let total = grading.total-points(qs)
  #let hard = grading.total-points(qs, filter: q => q.points >= 5)

  Total points: #total

  Points from hard questions: #hard
]

#q(points: 6)[
  == Hard Question

  #lorem(20)
]
```

Total points: 8

Points from hard questions: 6

Hard Question / 6

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

Question / 2

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

Once we have the total points of the text figured out, we need to define the grading key. Let's say the grades are in a three-grade system of "bad", "okay", and "good". We could define these grades like this:

```typ
#context [
  #let qs = question.all()
  #let total = grading.total-points(qs)

  #let grades = grading.grades(
    [bad], total * 2/4, [okay], total * 3/4, [good]
  )

  #grades
]
```

```
(
  (body: [bad], lower-limit: none, upper-limit: 4.0),
  (body: [okay], lower-limit: 4.0, upper-limit: 6.0),
  (body: [good], lower-limit: 6.0, upper-limit: none),
)
```

Hard Question / 6

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

Question / 2

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

Obviously we would not want to render this representation as-is, but `grading.grades()` gives us a convenient way to have all the necessary information, without assuming things like inclusive or exclusive point ranges. The `test.typ` example in the gallery has a more complete demonstration of a grading key.

IV Question templates and sample solutions

With the test structure out of the way, the next step is to actually write questions. There are endless ways of formulating questions, but some recurring formats come up regularly.

> *Note:* customizing the styles is currently very limited/not possible. I would be interested in changing this, so if you have ideas on how to achieve this, contact me and/or open a pull request. Until then, feel free to "customize using copy/paste".

Each question naturally has an answer, and producing sample solutions can be made very convenient if they are stored with the question right away. To facilitate this, this package provides

- `questions.solution`: this boolean state controls whether solutions are currently shown in the document.
- `questions.with-solution()`: this function sets the solution state temporarily, before switching back to the original state. The `small-example.typ` example in the gallery uses this to show an answered example question at the beginning of the document.

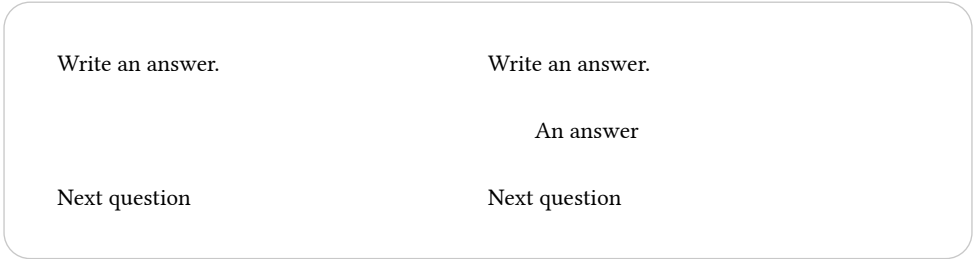
Additionally, the solution state can be set using the Typst CLI using `--input solution=true` (or `false`, which is already the default), or by regular state updates. Within context expressions, a question can use `questions.solution.get()` to find out whether solutions are shown. This is also used by Scrutinize's question templates.

Let's look at a free text question as a simple example:

IV.a Free text questions

In free text questions, the student simply has some free space in which to put their answer:

```typ
#import "@preview/scrutinize:0.2.0": grading, question, questions

#import questions: free-text-answer

// toggle the following comment or pass `--input solution=true`
// to produce a sample solution
// #questions.solution.update(true)

Write an answer.

#free-text-answer[An answer]

Next question
```



Left is the unanswered version, right the answered one. Note that the answer occupies the same space regardless of whether it is displayed or not, and that the height can also be overridden - see `questions.free-text-answer()`. The content of the answer is of course not limited to text.

IV.b single and multiple choice questions

These question types allow making a mark next to one or multiple choices. See `questions.single-choice()` and `questions.multiple-choice()` for details.

```typ
#import "@preview/scrutinize:0.2.0": grading, question, questions

#import questions: single-choice, multiple-choice

Which of these is the fourth answer?

#single-choice(
  range(1, 6).map(i => [Answer #i]),
  // 0-based indexing
  3,
)

Which of these answers are even?

#multiple-choice(
  range(1, 6).map(i => ([Answer #i], calc.even(i))),
)
```

Which of these is the fourth answer?

Answer 1	☐
Answer 2	☐
Answer 3	☐
Answer 4	☐
Answer 5	☐

Which of these answers are even?

Answer 1	☐
Answer 2	☐
Answer 3	☐
Answer 4	☐
Answer 5	☐

Which of these is the fourth answer?

Answer 1	☐
Answer 2	☐
Answer 3	☐
Answer 4	☒
Answer 5	☐

Which of these answers are even?

Answer 1	☐
Answer 2	☒
Answer 3	☐
Answer 4	☒
Answer 5	☐

V Module reference

V.a `scrutinize.question`

Variables:

q(body: `content`, ..args: `string`) -> `content`

Adds a question with its metadata, and renders it. The questions can later be accessed using the other functions in this module.

Parameters:

`body` (`content`) – the content to be displayed for this question

`..args` (`string`) – only named parameters: values to be added to the question's metadata

current() -> `dictionary`

Locates the most recently defined question; within a `q()` call, that is the question *currently* being defined.

This function is contextual and must appear within a `context` expression.

Example:

```typ
#context [
  #let points = question.current().points
  This question is worth #points points.

  I may award up to #(points + 1) points for great answers!
]
```

all() -> `array`

Locates all questions in the document, which can then be used to create grading keys etc.

This function is contextual and must appear within a `context` expression.

Example:

```typ
#context [
  #let qs = question.all()
  There are #qs.len() questions.

```

```
5   The first question is worth #qs.first().points points!
6 ]
```

counter `counter`

The question counter

Example:

```
1 #show heading: it => [Question #question.counter.display()]
```
typ

V.b `scrutinize.grading`

total-points(questions: `array`, filter: `function`) -> `integer`

Takes an array of question `metadata` dictionaries and returns the sum of their points. Note that the points metadata is optional and may therefore be `none`; if your test may contain questions without points, you have to take care of that.

This function also optionally takes a filter function. If given, the function will get the metadata of each question and must return a boolean.

Parameters:

`questions` (`array`) – an array of question `metadata` dictionaries

`filter` (`function` = `none`) – an optional filter function for determining which questions to sum up

grades(..args: `any`) -> `array`

A utility function for generating grades with upper and lower point limits. The parameters must alternate between grade names and threshold scores, with grades in ascending order. These will be combined in dictionaries for each grade with keys `body`, `lower-limit`, and `upper-limit`. The first (lowest) grade will have a `lower-limit` of `none`; the last (highest) grade will have an `upper-limit` of `none`.

Example:

```
1 #let total = 8
2 #let (bad, okay, good) = grading.grades(
3   [bad], total * 2/4, [okay], total * 3/4, [good]
4 )
5 [
6   You will need #okay.lower-limit points to pass,
7   everything below is a #bad.body grade.
8 ]
```
typ

Parameters:

`..args` (`any`) – only positional: any number of grade names interspersed with scores

V.c `scrutinize.questions`

- with-solution()
- free-text-answer()
- checkbox()
- multiple-choice()
- single-choice()

Variables:

- solution

with-solution(solution: `boolean` , body: `content`) -> `content`

Sets whether solutions are shown for a particular part of the document.

Parameters:

`solution` (`boolean`) – the solution state to apply for the body

`body` (`content`) – the content to show

free-text-answer(answer: `content` , height: `auto` `relative`) -> `content`

An answer to a free text question. If the document is not in solution mode, the answer is hidden but the height of the element is preserved.

Parameters:

`answer` (`content`) – the answer to (maybe) display

`height` (`auto` or `relative` = `auto`) – the height of the region where an answer can be written

checkbox(correct: `boolean`) -> `content`

A checkbox which can be ticked by the student. If the checkbox is a correct answer and the document is in solution mode, it will be ticked.

Parameters:

`correct` (`boolean`) – whether the checkbox is of a correct answer

multiple-choice(options: `array`) -> `content`

A table with multiple options that can each be true or false. Each option is a tuple consisting of content and a boolean for whether the option is correct or not.

Parameters:

`options` (`array`) – an array of (option, correct) pairs

single-choice(options: `array`, answer: `integer`) -> `content`

A table with multiple options of which one can be true or false. Each option is a content, and a second parameter specifies which option is correct.

Parameters:

`options` (`array`) – an array of contents

`answer` (`integer`) – the index of the correct answer, zero-based

solution `state`

A boolean state storing whether solutions should currently be shown in the document. This can be set using the Typst CLI using `--input solution=true` (or `false`, which is already the default) or by updating the state:

```typ
#questions.solution.update(true)
```

Additionally, `with-solution()` can be used to change the solution state temporarily.