Examination

v0.0.1

https://github.com/SillyFreak/typst-packages/examination

Clemens Koza

Abstract

Examination 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 2
II Questions and question metadata 2
III Grading 3
IV Question templates 4
V Sample solutions 4
VI Module reference 4

I Introduction

Examination has three general areas of focus:

- It provides a selection of question writing utilities, such as multiple choice or true/false questions.
- It helps with grading information: record the points that can be reached for each question and make them available for creating grading keys.
- 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:

```typst
#import "../src/lib.typ": grading, question

// you usually want to alias this, as you'll need it often
#let q = 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. Right now, the following metadata fields are available:

- `category`: an arbitrary category identifier (string) that can be used to group questions during further processing
- `points`: a number specifying how many points can be reached in that question
- additionally `body`: the complete content that was rendered as the question

The body is rendered as-is, but the two former 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:

```
#show heading: it => {
  // here, we need to access the current question's metadata
  question.current(q => [#it.body #h(1fr) / #q.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. Like Typst's `locate()` function, ordinarily, any computation has to happen inside as it can only return content – however, see the function's documentation for an escape hatch.

III Grading

The final puzzle piece is grading. There are many different possibilities to grade a test; Examination 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. Let's at the same time also look at question categories as a way to get subtotals:

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

  Total points: #total

  Points from hard questions: #hard
])

#q(category: "hard", 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:

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

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

  #grades
])
```

```
(
  (body: [bad], lower-limit: none, upper-limit: 4),
  (body: [okay], lower-limit: 4, upper-limit: 6),
  (body: [good], lower-limit: 6, 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

TODO

V Sample solutions

TODO

VI Module reference

VI.a `examination.question`

- q()
- current()
- all()

q(body: `content`, category: `string`, points: `integer`) -> `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

`category` (`string` = `none`) – an optional category to be added to the question's metadata

`points` (`integer` = `none`) – an optional point score to be added to the question's metadata

current(func: `function`, loc: `location`) -> `content | any`

Locates the most recently defined question; within a `q()` call, that is the question *currently* being defined. The located question's metadata (a dictionary) is used to call the provided function.

If the optional `loc` is provided, that location is used and the function's result is returned directly; otherwise, the result will be converted to a `content` by the necessary call to `locate`.

Example:

```
#question.current(q => [This question is worth #q.points points.])

#locate(loc => {
  let points = question.current(loc: loc, q => q.points)
  // note that `points` is an integer, not a content!
  let points-with-extra = points + 1
  // but eventually, `locate()` will convert to content
  [I may award up to #points-with-extra points for great answers!]
})
```

Parameters:

`func` (`function`) – a function that receives a metadata dictionary

`loc` (`location` = `none`) – if given, this is used to find the current question instead of `locate()`

all(func: `function`, loc: `location`) -> `content | any`

Locates all questions in the document, which can then be used to create grading keys etc. The array of located `metadata` objects is used to call the provided function; note that this is different from `current()`, where only the metadata dictionary is returned! The reasoning is that the locations of different questions may be useful for further queries, whereas for the current question having access to that location is not that useful. Use `map(q => q.value)` to only access the dictionaries.

If the optional `loc` is provided, that location is used and the function's result is returned directly; otherwise, the result will be converted to a `content` by the necessary call to `locate`.

Example:

```
#question.all(qs => [There are #qs.len() questions.])

#locate(loc => {
  let qs = question.all(loc: loc, qs => qs.map(q => q.value))
  // note that `qs` is an array, not a content!
  // but eventually, `locate()` will convert to content
  [The first question is worth #qs.first().points points!]
})
```

Parameters:

`func` (`function`) – a function that receives an array of `metadata` objects

`loc` (`location` = `none`) – if given, this is used to find the current question instead of `locate()`

counter `counter`

The question counter

Example:

```
#show heading.where(level: 2): it => [Question #question.counter.display()]
```

VI.b `examination.grading`

- total-points()
- grades()

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

Takes an array of question `metadata` objects (not 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 *dictionary* of each question and must return a boolean.

Parameters:

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

`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 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:

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

Parameters:

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