Markdown themes in practice

Vít Starý Novotný

Abstract

The Markdown package for TEX supports themes that allow TEXnicians to tailor the presentation of Markdown and YAML content on the page. In this article, I will show the current state of Markdown themes using the example of LATEX templates that I developed for the International Software Testing Qualifications Board (ISTQB). Readers will leave with actionable steps to create or modify Markdown themes for LATEX, and insights into extending these principles to other TEX engines.

Introduction

Although TEX has beautiful output, its input macro language is an acquired taste for many authors. The Markdown package for TEX allows authors to type familiar Markdown and YAML directly into a TEX document and receive a similarly beautiful output.

In my previous article, I introduced Markdown themes [5]. Much like CSS stylesheets, Markdown themes allow TEXnicians to tailor the presentation of Markdown and YAML content without complicating the document markup for authors. While that article used simple examples to explain the basic concepts behind Markdown themes, it did not demonstrate their application on a larger scale in real-world software projects.

In July 2023, I began working with the International Software Testing Qualifications Board (ISTQB) to help them typeset their certification study materials from Markdown and YAML sources. In this article, I discuss my work as a case study of using the Markdown package in a real-world software project.

Project overview

In my work, I developed a LATEX document class and six Markdown themes [1].

The LATEX document class is named `istqb` and it is stored in file `template/istqb.cls`. It implements the design of all ISTQB documents, defines the meaning of common Unicode characters, and defines LATEX markup such as `\istqbunnumberedsection`, `\istqblandscapebegin`, and `\istqblandscapeend`.

The Markdown themes are named `istqb/*` and stored in files `template/markdowntheme*.tex` and `*.sty`; see also Figure 1. Here is what they do:

- The theme `istqb/common` enables Markdown syntax extensions, implements the loading of YAML language definitions and document metadata into TEX macros, and defines the mapping between Markdown elements and LATEX markup. The remaining themes are based on this theme and they implement support for specific types of ISTQB documents.
- The `istqb/body-of-knowledge` and `syllabus` themes are used in ISTQB Body of Knowledge and Syllabus documents. At the time of writing, the themes implement no extra functionality.
- The theme `istqb/sample-exam` implements the loading of YAML question definitions into TEX macros in ISTQB Sample Exam Questions and Answers documents. The following two themes are based on this theme.
- The theme `istqb/sample-exam/questions` implements the typesetting of questions in ISTQB Sample Exam Questions documents.
- The theme `istqb/sample-exam/answers` implements typesetting of answer keys and answers in ISTQB Sample Exam Answers documents.

In the rest of this article, I show the main concepts behind Markdown themes using the examples of ISTQB Sample Exam Questions and Answers documents, which use the themes `istqb/sample-exam/questions` and `/answers`.

With Markdown themes, your document can wear many different disguises, just like the wolf.

doi.org/10.47397/tb/45-2/tb140starynovotny-markdown-themes

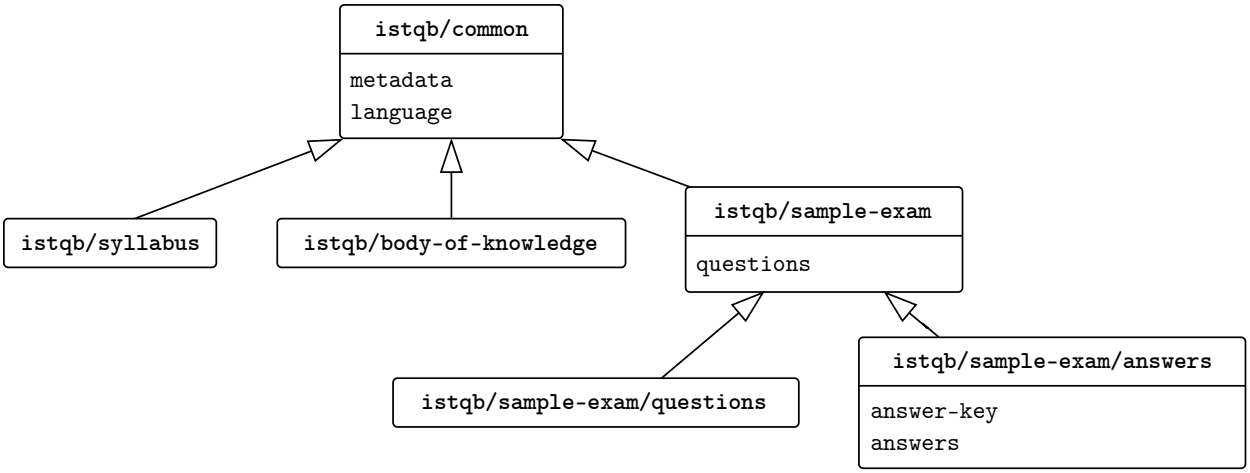
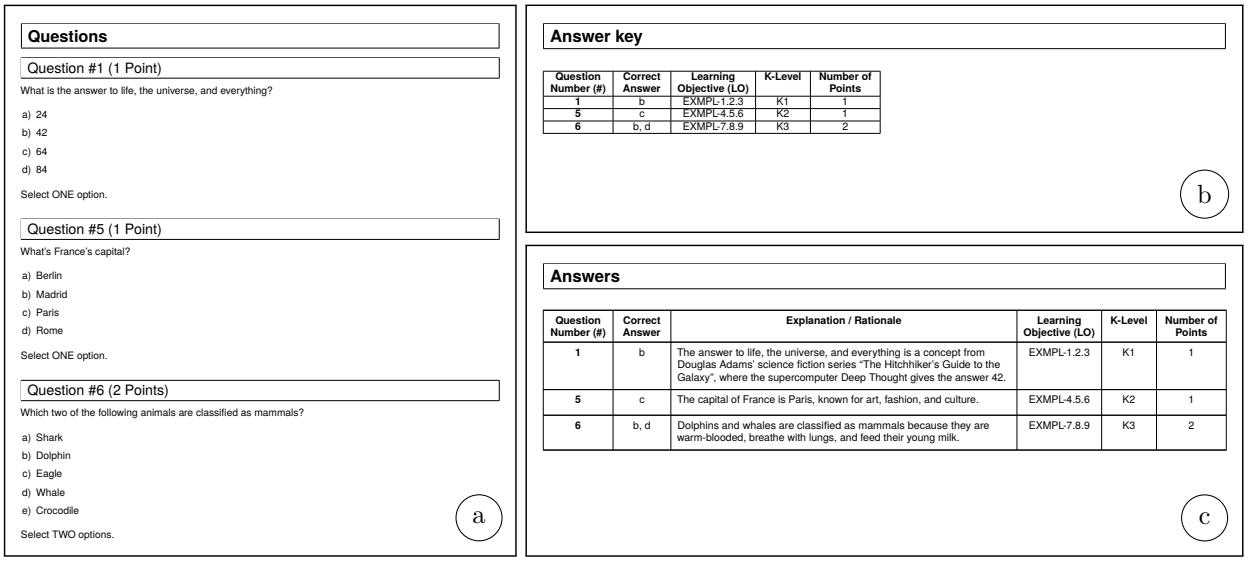


Figure 1: A class diagram of the six Markdown themes that I developed for the International Software Testing Qualifications Board (ISTQB). The snippets `metadata`, `language`, `questions`, `answer-key`, and `answers` specify the public interface of the themes and arrows specify inheritance.



Questions

Question #1 (1 Point)

What is the answer to life, the universe, and everything?

a) 24
b) 42
c) 64
d) 84

Select ONE option.

Question #5 (1 Point)

What's France's capital?

a) Berlin
b) Madrid
c) Paris
d) Rome

Select ONE option.

Question #6 (2 Points)

Which two of the following animals are classified as mammals?

a) Shark
b) Dolphin
c) Eagle
d) Whale
e) Crocodile

Select TWO options.

a

Answer key

Question Number (#)	Correct Answer	Learning Objective (LO)	K-Level	Number of Points
1	b	EXMPL-1.2.3	K1	1
5	c	EXMPL-4.5.6	K2	1
6	b, d	EXMPL-7.8.9	K3	2

b

Answers

Question Number (#)	Correct Answer	Explanation / Rationale	Learning Objective (LO)	K-Level	Number of Points
1	b	The answer to life, the universe, and everything is a concept from Douglas Adams' science fiction series "The Hitchhiker's Guide to the Galaxy", where the supercomputer Deep Thought gives the answer 42.	EXMPL-1.2.3	K1	1
5	c	The capital of France is Paris, known for art, fashion, and culture.	EXMPL-4.5.6	K2	1
6	b, d	Dolphins and whales are classified as mammals because they are warm-blooded, breathe with lungs, and feed their young milk.	EXMPL-7.8.9	K3	2

c

Figure 2: Three different ways to typeset question definitions in ISTQB Sample Exam Questions and Answers documents: a) a list of questions, b) an answer key, and c) a list of answers.

1 Question definitions

As an example of question definitions, I use the following YAML file named `questions.yml`:

```
num-questions: 3
max-score: 4
pass-score: 50  # percent
duration: [10, 15]  # minutes
questions:
  1:
    learning-objective: 1.2.3
    k-level: K1
    number-of-points: 1
    question: >
      What is the answer to life,
      the universe, and everything?
    answers: {a: 24, b: 42, c: 64, d: 84}
    correct: [b]
    explanation: >
      The answer to life, the universe,
      and everything is a concept from
      Douglas Adams' science fiction
      series "The Hitchhiker's Guide to
      the Galaxy", where the supercomputer
      Deep Thought gives the answer 42.
  5:
    learning-objective: 4.5.6
    k-level: K2
    number-of-points: 1
    question: What's France's capital?
    answers: {a: Berlin, b: Madrid,
              c: Paris, d: Rome}
    correct: [c]
    explanation: >
      The capital of France is Paris,
      known for art, fashion, and culture.
  6:
    learning-objective: 7.8.9
    k-level: K3
    number-of-points: 2
    question: >
      Which two of the following animals
      are classified as mammals?
    answers: {a: Shark, b: Dolphin,
              c: Eagle, d: Whale,
              e: Crocodile}
    correct: [b, d]
    explanation: >
      Dolphins and whales are classified
      as mammals because they are
      warm-blooded, breathe with lungs,
      and feed their young milk.
```

The file specifies three questions. For each question, it provides up to five possible answers.

2 User interface

In this section, I show how we can use themes `istqb/`
`/sample-exam/questions`, and `/answers` to typeset the question definitions from the previous section.

2.1 Typesetting questions

As an example of an ISTQB Sample Exam Questions document, I use the following LaTeX file:

```
\documentclass{istqb}
\usepackage{markdown}
\markdownSetup {
  import = {
    istqb/sample-exam/questions =
      questions as qst
  }
}
\begin{document}
\istqbunnumberedsection{Questions}
\markdownInput[snippet=qst]{questions.yml}
\end{document}
```

The file imports the snippet `questions` from theme `istqb/sample-exam/questions` and uses it to:

1. Process question definitions in `questions.yml`.
2. Typeset the list of questions shown in Figure 2a.

2.2 Typesetting answer key and answers

As an example of an ISTQB Sample Exam Answers document, I use the following LaTeX file:

```
\documentclass{istqb}
\usepackage{markdown}
\markdownSetup {
  import = {
    istqb/sample-exam/answers = {
      answer-key as key,
      answers as ans,
    },
  }
}
\begin{document}
\istqblandscapebegin
\istqbunnumberedsection{Answer key}
\markdownInput[snippet=key]{questions.yml}
\istqbunnumberedsection{Answers}
\markdownInput[snippet=ans]{questions.yml}
\istqblandscapeend
\end{document}
```

The file imports snippets `answers` and `answer-key` from theme `istqb/sample-exam/answers` and uses them to:

1. Process question definitions in `questions.yml`.
2. Typeset the answer key shown in Figure 2b.
3. Typeset the list of answers shown in Figure 2c.

3 Implementation

In this section, I show the implementation of ISTQB Sample Exam Questions and Answers documents. To make programming easier, I use the high-level expl3 language in addition to plain TEX and LATEX 2_ε.

3.1 Processing question definitions

Both the snippet `questions` from the theme `istqb/sample-exam/questions` and the snippet `answers` from the theme `/answers` process question definitions before typesetting them. For the processing, they use the snippet `questions` from the theme `istqb/sample-exam`, which I describe in this section.

First, I define a key–value `istqb/questions`:

```
\keys_define:nn
  { istqb / questions }
  { num-questions .int_gset:N =
      \g_istqb_num_questions_int,
    max-score .int_gset:N =
      \g_istqb_max_score_int,
    pass-score .int_gset:N =
      \g_istqb_pass_score_int }
```

The key–value stores the values in top-level unstructured fields `num-questions`, `max-score`, and `pass-score` from question definitions to variables.

Next, I define a key–value `istqb/questions//duration`:

```
\keys_define:nn
  { istqb / questions / duration }
  { 1 .int_gset:N =
      \g_istqb_duration_min_int,
    2 .int_gset:N =
      \g_istqb_duration_max_int }
```

The key–value stores the values in the top-level structured field `duration` to variables.

Then, I define the snippet `questions` itself:

```
\seq_new:N \g_istqb_questions_seq
\markdownSetupSnippet
  { questions }
  { jekyllData,
    expectJekyllData,
    renderers = {
      jekyllDataBegin = {
        \seq_gclear:N
          \g_istqb_questions_seq },
      jekyllData(String|Number) = {
        \keys_set:nn
          { istqb / questions }
          { { #1 } = { #2 } }},
      jekyllDataMappingBegin = ,
      jekyllDataSequenceBegin = {
        \str_case:nn
          { #1 }
          { { duration } {
              \markdownSetup
                { code = \group_begin:,
                  renderers = {
                    jekyllData(String
                               |Number) = {
                      \keys_set:nn
                        { istqb / questions /
                          duration }
                        {{ ##1 } = { ##2 }}},
                    jekyllDataSequenceEnd =
                      \group_end: }}}}},
      jekyllData(Mapping|Sequence)Begin += {
        \str_case:nn
          { #1 }
          { { questions } {
              \markdownSetup
                { code = \group_begin:,
                  renderers = {
                    jekyllData(Mapping
                               |Sequence)End =
                  },
                  snippet = istqb
                    / sample-exam / questions
                    / list,
                  renderers = {
                    jekyllData(Mapping
                               |Sequence)End
                      += \group_end: }}}}}}}
```

The snippet processes question definitions as follows:

1. Define an empty sequence that will store question numbers.
2. Pass unstructured top-level fields to the key–value `istqb/questions`.
3. Pass the structured top-level field `duration` to the key–value `istqb/questions/duration`.
4. Pass the structured top-level field `questions` to a snippet `questions/list`.

Next, I define the snippet `questions/list`:

```
\markdownSetupSnippet
  { questions / list }
  { renderers = {
      jekyllDataMappingBegin = {
        \group_begin:
        \tl_set:Nn
          \l_istqb_current_question_tl
          { #1 }
        \seq_gput_right:NV
          \g_istqb_questions_seq
          \l_istqb_current_question_tl
        \markdownSetup
          { renderers = {
              jekyllDataMappingEnd = },
            snippet = istqb / sample-exam
              / questions / *,
            renderers = {
              jekyllDataMappingEnd +=
                \group_end: }}}}}
```

The snippet processes each question as follows:

1. Store the current question number.
2. Pass all fields to a snippet `questions/*`.

Then, I define key–value `istqb/questions/*`:

```
\prop_new:N
  \g_istqb_question_learning_objective_prop
\prop_new:N
  \g_istqb_question_k_level_prop
\prop_new:N
  \g_istqb_question_number_of_points_prop
\prop_new:N
  \g_istqb_question_text_prop
\prop_new:N
  \g_istqb_question_explanation_prop
\keys_define:nn
  { istqb / questions / * }
  { learning-objective .code:n = {
      \prop_gput:cVn
        { g_istqb_question_learning_objective
          _prop }
        \l_istqb_current_question_tl
        { #1 } },
    k-level .code:n = {
      \prop_gput:NVn
        \g_istqb_question_k_level_prop
        \l_istqb_current_question_tl
        { #1 } },
    number-of-points .code:n = {
      \prop_gput:cVn
        { g_istqb_question_number_of_points
          _prop }
        \l_istqb_current_question_tl
        { #1 } },
    question .code:n = {
      \prop_gput:NVn
        \g_istqb_question_text_prop
        \l_istqb_current_question_tl
        { #1 } },
    explanation .code:n = {
      \prop_gput:NVn
        \g_istqb_question_explanation_prop
        \l_istqb_current_question_tl
        { #1 } }}
```

The key–value stores the values in unstructured fields `number-of-points`, `learning-objective`, `k--level`, `explanation`, and `question` to dicts. The dicts use the current question number as the key.

Next, I define the snippet `questions/*`:

```
\markdownSetupSnippet
  { questions / * }
  { renderers = {
      jekyllData(String|Number) = {
        \keys_set:nn
          { istqb / questions / * }
          { { #1 } = { #2 } }},
      jekyllDataSequenceBegin = {
        \str_case:nn
          { #1 }
          { { correct } {
              \markdownSetup
                { code = \group_begin:,
                  renderers = {
                    jekyllDataSequenceEnd =
                  },
                  snippet = istqb
                    / sample-exam / questions
                    / * / correct,
                  renderers = {
                    jekyllDataSequenceEnd +=
                      \group_end: }}}}},
      jekyllDataMappingBegin = {
        \str_case:nn
          { #1 }
          { { answers } {
              \markdownSetup
                { code = \group_begin:,
                  renderers = {
                    jekyllDataMappingEnd = },
                  snippet = istqb
                    / sample-exam / questions
                    / * / answers,
                  renderers = {
                    jekyllDataMappingEnd +=
                      \group_end: }}}}}}
```

The snippet processes question definitions as follows:

1. Pass unstructured fields to the key–value `istqb//questions/*`.
2. Pass the structured field `correct` to a snippet `questions/*/correct`.
3. Pass the structured field `answers` to a snippet `questions/*/answers`.

Notice the design pattern on lines 44–60, 64–79, and 126–154 that locally applies a ⟨*snippet*⟩ to an ⟨*element*⟩.[1] This pattern redefines the renderer ⟨*element*⟩`Begin`, which is placed to the output when the ⟨*element*⟩ starts, as follows:

1. Open a TEX group and apply the ⟨*snippet*⟩.
2. Redefine the renderer ⟨*element*⟩`End`, which is placed to the output when the ⟨*element*⟩ ends, so that it closes the TEX group.

Finally, I define snippets `questions/*/answers` and `/correct`:

```
\prop_new:N \g_istqb_answer_keys_prop
\prop_new:N \g_istqb_answers_prop
\seq_new:N \l_istqb_current_answer_keys_seq
\markdownSetupSnippet
  { questions / * / answers }
```

[1] Such design patterns can be repetitive and difficult to understand without additional comments in the code. Markdown Enhancement Proposal (MEP) 445 [6] envisions support for higher-order snippets that would make it possible to hide such design patterns behind easy-to-read shorthands.

```
  { renderers = {
      jekyllData(String|Number) = {
        \seq_put_right:Nn
          \l_istqb_current_answer_keys_seq
          { #1 }
        \tl_set:NV
          \l_tmpa_tl
          \l_istqb_current_question_tl
        \tl_put_right:Nn
          \l_tmpa_tl
          { / #1 }
        \prop_gput:NVn
          \g_istqb_answers_prop
          \l_tmpa_tl
          { #2 } },
      jekyllDataMappingEnd += {
        \clist_set_from_seq:NN
          \l_istqb_current_answer_keys_clist
          \l_istqb_current_answer_keys_seq
        \prop_gput:NVv
          \g_istqb_answer_keys_prop
          \l_istqb_current_question_tl
          { l_istqb_current_answer_keys
            _clist } }}}
\prop_new:N \g_istqb_answer_correct_keys_prop
\seq_new:N
  \l_istqb_current_answer_correct_keys_seq
\markdownSetupSnippet
  { questions / * / correct }
  { renderers = {
      jekyllData(String|Number) = {
        \seq_put_right:cn
          { l_istqb_current_answer_correct
            _keys_seq }
          { #2 } },
      jekyllDataSequenceEnd += {
        \clist_set_from_seq:cc
          { l_istqb_current_answer_correct
            _keys_clist }
          { l_istqb_current_answer_correct
            _keys_seq }
        \prop_gput:NVv
          \g_istqb_answer_correct_keys_prop
          \l_istqb_current_question_tl
          { l_istqb_current_answer_correct
            _keys_clist } }}}
```

The snippets accumulate potential and correct answer letters in a sequence, respectively. Then, they store the sequence as a comma-list to a dict that uses the current question number as the key.

Moreover, the snippet `questions/*/answers` stores potential answer texts to a dict that uses ⟨*current question number*⟩`/`⟨*answer letter*⟩ as key.

Notice that I used no format-specific code in this section. Therefore, I can use the theme `istqb/sample-exam` with any format that supports expl3 such as plain TEX and ConTEXt, not just with LATEX.

3.2 Typesetting questions

In this section, I describe the snippet `questions` from theme `istqb/sample-exam/questions`. This snippet typesets the list of questions in Figure 2a.

First, I import the theme `istqb/sample-exam` and I use the snippet `questions` from this theme to process question definitions:

```
\markdownSetup
  { import = istqb / sample-exam }
\markdownSetupSnippet
  { questions }
  { snippet = istqb / sample-exam
      / questions,
```

After the question definitions have been processed, I iterate over all question numbers. For each question number, I define a variable with code that typesets the corresponding question:

```
    renderers = {
      jekyllDataEnd = {
        \seq_map_inline:Nn
          \g_istqb_questions_seq
          { \tl_set:Nn
              \l_istqb_question_tl
              {
```

First, I add a section heading for the question:

```
                \tl_set:Nn
                  \l_tmpa_tl
                  { Question~\# ##1~( }
                \prop_get:cnN
                  { g_istqb_question_number
                    _of_points_prop }
                  { ##1 }
                  \l_tmpb_tl
                \tl_put_right:NV
                  \l_tmpa_tl
                  \l_tmpb_tl
                \tl_put_right:Nn
                  \l_tmpa_tl
                  { ~Point }
                \int_compare:VNnF
                  \l_tmpb_tl = { 1 }
                  { \tl_put_right:Nn
                      \l_tmpa_tl
                      { s } }
                \tl_put_right:Nn
                  \l_tmpa_tl
                  { ) }
                \exp_args:NNV
                  \subsection *
                  \l_tmpa_tl
                \exp_args:NVV
                  \markboth
                  \l_tmpa_tl
                  \l_tmpa_tl
                \exp_args:NnnV
                  \addcontentsline
```

```
                { toc }
                { subsection }
                \l_tmpa_tl
```

Next, I add the question text and potential answers:

```
              \prop_item:Nn
                \g_istqb_question_text_prop
                { ##1 }
              \prop_get:NnN
                \g_istqb_answer_keys_prop
                { ##1 }
                \l_tmpa_clist
              \begin { enumerate }
              \clist_map_inline:Nn
                \l_tmpa_clist
                { \item [ ####1 ) ]
                    \prop_item:Nn
                      \g_istqb_answers_prop
                      { ##1 / ####1 } }
              \end { enumerate }
              \medskip
```

Lastly, I add the text "Select ⟨*number of correct answers*⟩ option(s).":

```
              \prop_get:cnN
                { g_istqb_answer_correct
                  _keys_prop }
                { ##1 }
                \l_tmpa_clist
              \int_set:Nn
                \l_tmpa_int
                { \clist_count:N
                    \l_tmpa_clist }
              Select~\int_case:nn
                { \l_tmpa_int }
                { { 1 } { ONE~option }
                  { 2 } { TWO~options } }
            }
```

Finally, I typeset the code from the variable at natural height and store the result to a vertical box:

```
          \vbox_set:NV
            \l_tmpa_box
            \l_istqb_question_tl
```

For short questions, I insert the box to the current list for typesetting to prevent page breaks within the question. For longer questions, I place the content of the variable to the input stream, so that page breaks can occur naturally:

```
          \dim_compare:nNnTF
            { \box_ht:N \l_tmpa_box }
            >
            { 0.3 \paperheight }
            { \tl_use:N
                \l_istqb_question_tl }
            { \box_use:N \l_tmpa_box }
          \par }}}}
```

3.3 Typesetting the answer key

In this section, I describe the snippet `answer-key` from the theme `istqb/sample-exam/answers`. This snippet typesets the answer key in Figure 2b.

First, I load packages multicol and supertabular:

```
\RequirePackage { multicol }
\RequirePackage { supertabular }
\RequirePackage { array }
\newcolumntype
  { C }
  [ 1 ]
  { >{ \centering\arraybackslash } p { #1 } }
```

The packages allow me to typeset the answer key as a table in a two-column layout that automatically inserts column breaks.

Next, I import the theme `istqb/sample-exam` and I use the snippet `questions` from this theme to process question definitions:

```
\markdownSetup
  { import = istqb / sample-exam }
\markdownSetupSnippet
  { answer-key }
  { snippet = istqb / sample-exam
      / questions,
```

After the question definitions have been processed, I start a two-column layout:

```
    renderers = {
      jekyllDataEnd = {
        \begin { multicols } { 2 }
```

Then, I set the heading and the tail of the table:

```
          \tablehead
            { \hline
              \textbf
                { Question~Number~(\#) } &
              \textbf
                { Correct~Answer } &
              \textbf
                { Learning~Objective~(LO) } &
              \textbf
                { K-Level } &
              \textbf
                { Number~of~Points } \\ }
          \tabletail { \hline }
          \tablelasttail { \hline }
```

Next, I define a variable that typesets the table:

```
          \tl_set:Nn
            \l_istqb_answer_key_table_tl
            {
```

First, I start the table:

```
              \begin
                { supertabular }
                { | C { 1.9cm } | C { 1.5cm }
                  | C { 2.4cm } | C { 1.4cm }
                  | C { 1.9cm } | } }
```

Next, I iterate over all question numbers:

```
      \seq_map_inline:Nn
        \g_istqb_questions_seq
        {
          \tl_put_right:Nn
            \l_istqb_answer_key_table_tl
            { \hline }
```

For each question, I add the question number:

```
          \tl_put_right:Nn
            \l_istqb_answer_key_table_tl
            { \textbf { ##1 } & }
```

Next, I add the correct answer letters:

```
          \prop_get:cnN
            { g_istqb_answer_correct
              _keys_prop }
            { ##1 }
            \l_tmpa_clist
          \tl_put_right:Ne
            \l_istqb_answer_key_table_tl
            { \clist_use:Nn
                \l_tmpa_clist
                { ,~ } & }
```

Then, I add the learning objective:

```
          \tl_put_right:NV
            \l_istqb_answer_key_table_tl
            \g_istqb_prefix_tl
          \tl_put_right:Nn
            \l_istqb_answer_key_table_tl
            { - }
          \prop_get:cnN
            { g_istqb_question_learning
              _objective_prop }
            { ##1 }
            \l_tmpa_tl
          \tl_put_right:NV
            \l_istqb_answer_key_table_tl
            \l_tmpa_tl
          \tl_put_right:Nn
            \l_istqb_answer_key_table_tl
            { & }
```

Next, I add the K-level:

```
          \prop_get:NnN
            \g_istqb_question_k_level_prop
            { ##1 }
            \l_tmpa_tl
          \tl_put_right:NV
            \l_istqb_answer_key_table_tl
            \l_tmpa_tl
          \tl_put_right:Nn
            \l_istqb_answer_key_table_tl
            { & }
```

Lastly, I add the number of points:

```
          \prop_get:cnN
            { g_istqb_question_number
              _of_points_prop }
            { ##1 }
            \l_tmpa_tl
          \tl_put_right:NV
            \l_istqb_answer_key_table_tl
            \l_tmpa_tl
          \tl_put_right:Nn
            \l_istqb_answer_key_table_tl
            { \\ }
        }
```

After I have iterated over all question numbers, I end the table, I place the content of the variable to the input stream, and I end the multicolumn layout:

```
      \tl_put_right:Nn
        \l_istqb_answer_key_table_tl
        { \end { supertabular } }
      \tl_use:N
        \l_istqb_answer_key_table_tl
      \end { multicols } }}}
```

3.4 Typesetting answers

In this section, I describe the snippet `answers` from the theme `istqb/sample-exam/answers`. This snippet typesets the list of answers in Figure 2c.

First, I load package longtable:

```
\RequirePackage { longtable }
\dim_const:Nn
  \c_explanation_width_dim
  { 11.15cm }
```

The package allows me to typeset the list of answers as a table that automatically inserts page breaks.

Next, I use the snippet `questions` from theme `istqb/sample-exam` to process question definitions:

```
\markdownSetupSnippet
  { answers }
  { snippet = istqb / sample-exam
      / questions,
```

After the question definitions have been processed, I define a variable that typesets the table:

```
    renderers = {
      jekyllDataEnd = {
        \group_begin:
        \tl_set:Nn
          \l_istqb_answers_table_tl
          {
```

First, I start the table and I set its heading:

```
            \begin
              { longtable }
              { | C { 1.9cm } | C { 1.5cm }
                | p
                { \c_explanation_width_dim }
                | C { 2.4cm } | C { 1.4cm }
                | C { 1.9cm } | }
            \hline
            \textbf
              { Question~Number~(\#) } &
            \textbf { Correct~Answer } &
```

```
            \multicolumn
              { 1 }
              { C
                { \c_explanation_width_dim }
                | }
              { \textbf
                  { Explanation~/~Rationale }
              } &
            \textbf
              { Learning~Objective~(LO) } &
            \textbf { K-Level } &
            \textbf { Number~of~Points } \\
            \hline
            \endhead }
```

Next, I iterate over all question numbers:

```
        \seq_map_inline:Nn
          \g_istqb_questions_seq
          {
```

For each question, I add the question number:

```
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { \textbf
                  { ##1 }
                \addcontentsline
                  { toc }
                  { subsection }
                  { Question~\# ##1 } & }
```

Next, I add the correct answer letters:

```
            \prop_get:cnN
              { g_istqb_answer_correct
                _keys_prop }
              { ##1 }
              \l_tmpa_clist
            \tl_put_right:Ne
              \l_istqb_answers_table_tl
              { \clist_use:Nn
                  \l_tmpa_clist
                  { ,~ } & }
```

Then I add the explanation text:

```
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { \begin
                  { minipage }
                  [ t ]
                  \c_explanation_width_dim }
            \prop_get:cnN
              { g_istqb_question_explanation
                _prop }
              { ##1 }
              \l_tmpa_tl
            \tl_put_right:NV
              \l_istqb_answers_table_tl
              \l_tmpa_tl
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { \end { minipage }
                \medskip }
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { & }
```

Next, I add the learning objective:

```
            \tl_put_right:NV
              \l_istqb_answers_table_tl
              \g_istqb_prefix_tl
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { - }
            \prop_get:cnN
              { g_istqb_question_learning
                _objective_prop }
              { ##1 }
              \l_tmpa_tl
            \tl_put_right:NV
              \l_istqb_answers_table_tl
              \l_tmpa_tl
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { & }
```

Then, I add the K-level:

```
            \prop_get:NnN
              \g_istqb_question_k_level_prop
              { ##1 }
              \l_tmpa_tl
            \tl_put_right:NV
              \l_istqb_answers_table_tl
              \l_tmpa_tl
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { & }
```

Lastly, I add the number of points:

```
            \prop_get:cnN
              { g_istqb_question_number_of
                _points_prop }
              { ##1 }
              \l_tmpa_tl
            \tl_put_right:NV
              \l_istqb_answers_table_tl
              \l_tmpa_tl
            \tl_put_right:Nn
              \l_istqb_answers_table_tl
              { \\ \hline } }
```

After I have iterated over all question numbers, I end the table and I place the content of the variable to the input stream:

```
        \tl_put_right:Nn
          \l_istqb_answers_table_tl
          { \end { longtable } }
        \tl_use:N
          \l_istqb_answers_table_tl
        \group_end: }}}
```

Conclusion

In this article, I have demonstrated the practical application of Markdown themes through a project that enabled the International Software Testing Qualifications Board (ISTQB) to produce their certification study materials from Markdown and YAML sources. While my previous article [5] focused on the underlying concepts of Markdown themes, this article provides concrete code used in a real-world software project. I hope this practical demonstration raises awareness of Markdown themes and illustrates how users can incorporate them into their own projects.

For ISTQB, the project has yielded numerous benefits: Writing text in a structured format using Markdown and YAML, while generating visually appealing outputs with LaTeX, facilitates the separation of content from formatting. This ensures consistent application of the document's visual style across all ISTQB content. Additionally, the structured text enables content verification against YAML schemas and ISTQB writing rules and allows for the creation of a complex knowledge base through automated processing. This enhances the quality of learning materials and reduces administrative overhead.

Moreover, the plain text formats of Markdown and YAML offer significant advantages over binary formats like Microsoft Office. They allow for efficient version control, better tracking of changes, collaborative editing, and fewer defects in the final products. The capability to produce various output formats, such as EPUB, HTML, and PDF with functional hyperlinks and cross-references, further amplifies the utility of this approach.

Related work

In my approach, I developed an event-based LaTeX parser that constructs and typesets expl3 data structures that represent YAML files.[2] My approach works in any TeX engine with shell access, such as pdfTeX and XeTeX, not just LuaTeX.

In the previous issue of *TUGboat* [4], Erik Nijenhuis showed a different approach towards typesetting YAML files in LaTeX. In their approach, Erik used their lua-placeholders library [3] to load YAML files into Lua tables and then query them from TeX code. Erik's approach requires LuaTeX but can be more convenient for non-programmers.

Both Erik's and my approaches use the tinyyaml Lua library [2]. LuaTeX users who are interested in processing YAML files directly from Lua code may find it convenient to use tinyyaml directly.

Acknowledgements

I wish to extend my special thanks to Tereza Vrabcová, Marei Peischl, Daniel Poľan, and Petr Sojka for their invaluable insights and thorough review of my work. Their expertise and thoughtful feedback have been instrumental in shaping the final manuscript.

I would also like to thank Greg at `fiverr.com/quickcartoon` for their illustrations of the wolf mascot, which have provided an engaging visual identity of the Markdown package over the past four years.

References

[1] ISTQB.ORG. LaTeX+Markdown template, 2024. `github.com/istqborg/istqb_product_base`

[2] Z. Lee. lua-tinyyaml: A tiny YAML (subset) parser for pure Lua, 2023. `ctan.org/pkg/lua-tinyyaml`

[3] E. Nijenhuis. lua-placeholders: Specifying placeholders for demonstration purposes, 2024. `ctan.org/pkg/lua-placeholders`

[4] E. Nijenhuis. Specifying and populating documents in YAML with lua-placeholders in LaTeX. *TUGboat* 45(1):65–76, 2024. `doi.org/10.47397/tb/45-1/tb139nijenhuis-placeholders`

[5] V. Novotný. Markdown 2.10.0: LaTeX themes & snippets, two flavors of comments, and luametaTeX. *TUGboat* 42(2):186–193, 2021. `doi.org/10.47397/tb/42-2/tb131novotny-markdown`

[6] V. Starý Novotný. Parametric snippets, 2024. `github.com/Witiko/markdown/issues/445`

◇ Vít Starý Novotný
Studená 453/15
Brno 63800, Czech Republic
`witiko (at) mail dot muni dot cz`
`github.com/witiko`

[2] My focus on processing and typesetting *YAML* files may seem contrary to the title of this article "*Markdown* themes in practice". However, authors may use Markdown markup in YAML files. In the examples from this article, we might use Markdown to format questions, answers, and explanations.