Class 4: Creating Content for Slides and Handouts

Table of Contents / Agenda

1 Creating Slides and Handouts: Introduction

Content is created using the Org markup language. General information on how to use the markeup language can be found in the Org manual. Hence, we focus on common slide- and handout-related tasks that cannot be achieved with common Org markup language. These tasks require either raw HTML tags or LaTeX code, or custom Emacs lisp code. For tasks that require custom Emacs Lisp code, users can put code snippets we present here into their Emacs init file.

> We use reveal.js as the default slide format, and while Beamer slides are not completely supported by Org-Coursepack yet, in some cases we do provide the same functionality for Beamer as well as reveal.js. In those cases we have a note describing how to achieve the functionality in Beamer.

2 Using Raw HTML Tags and LaTeX Code

Directly quoting raw HTML tags and LaTeX code allows users to have granular control over how contents are presented. Such quotes will be only included in their corresponding outputs that are HTML-based (e.g., reveal.js) and LaTeX-based, respectively. Hence, to understand the information that follow, the readers should be familiar with the information in the Org manual on Quoting HTML tags and Quoting LaTeX code.

Inline Raw Code

To use Org macros with raw LaTeX code (e.g., surround a macro with LaTeX code), use `@@latex:your code here@@` (same grammar applies to HTML as well) like the following:

```
@@latex:{\small@@ {{{COURSE}}} @@latex:}@@
```

3 Features for Both Slides and Handouts

3.1 Specifying Attributes

Org mode allows users to specify attributes to raw HTML tags or LaTeX code using `#+ATTR_FORMAT:` grammar. For example, the following shows how to specify the width of an image.

```
#+ATTR_HTML: :width 80%
[[/img/image.png]]
```

For more information, see the tutorial on Images and XHTML export.

3.2 Changing Font Sizes

One of the frequently used use cases of raw HTML or LaTeX code in Org markup is changing the font size of a specific text.

For example, to apply a smaller font size in HTML outputs, the user can use the following code.

```
#+HTML: <span style=font-size:20pt>
Content with smaller font
#+HTML: </span>
```

In LaTeX handouts, the user can use the code below.

```
#+LATEX: {\small
Content with smaller font
#+LATEX: }
```

Since raw code that is irrelevant to the specific output format (e.g., HTML codes in a LaTeX output) will be ignored, users can safely combine HTML and LateX codes and use them together like so:

```
#+LATEX: {\small
#+HTML: <span style=font-size:20pt>
Content with smaller font
#+HTML: </span>
#+LATEX: }
```

3.3 Using a Dummy Heading

Instructors may want the option to present content of a tree without its heading. To do so, follow the instructions at `https://orgmode.org/worg/org-hacks.html#ignoreheadline`. Specifically, include the following in your init file, and any header with the `:ignore:` tag will not be printed in exported outputs.

```
(require 'ox-extra)
(ox-extras-activate '(ignore-headlines))
```

4 Slides: Features for reveal.js

> Note that reveal.js is HTML-based, so any raw HTML tags (e.g., via `#+HTML:`) or attributes (e.g., via `#+ATTR_HTML:`) will be applied to reveal.js as well as all HTML-based output formats. For codes that are only for reveal.js, one should use `#+REVEAL:` (`#+ATTR_REVEAL:`) instead of `#+HTML:` (`#+ATTR_HTML:`) to avoid unnecessary tags being exported.

4.1 List Fragments

One can easily obtain list fragments (make items in the list appear sequentially) using reveal.js. Simply add `#+ATTR_REVEAL: :frag (appear)` before the list. See the example below.

```
#+ATTR_REVEAL: :frag (appear)
- I appear first.
- I appear second.
- I appear third.
```

> Similarly, a list fragment can be obtained on the Beamer output by including `#+ATTR_BEAMER: :overlay <+->` before the list.

4.2 Splitting slides

To split content into multiple slides, insert the following code between the areas where you want the split to happen.

```
#+REVEAL: split
```

> Similarly, a frame break can be inserted in Beamer by using `#+BEAMER: \framebreak`.

4.3 Embedding Youtube videos

One can use the following example to embed a YouTube video in a slide. The example specifies at which points of the video the viewing will start (1 second in) and end (60 seconds in).

```
#+BEGIN_EXPORT HTML
<iframe width="1066" height="570"
src="https://www.youtube.com/embed/SzA2YODtgK4?start=01&end=60" allowfullscreen>
</iframe>
#+END_EXPORT
```

4.4 Speaker Notes

An instructor may create a speaker note that accompanies a lecture slide. reveal.js will display the speaker note in a separate browser window. To create a speaker note, use a `NOTES` block as shown in the example below.

```
#+BEGIN_NOTES
- This is a speaker note.
#+END_NOTES
```

The following code needs to be inserted in the init file to hide speaker notes in LaTeX, reStructuredText, and HTML output formats.

> Note that using the example code below will also make speaker notes appear properly on Beamer.

```
(defun my/process-NOTES-blocks (text backend info)
  "Filter NOTES special blocks in export."
  (cond
   ((eq backend 'latex)
    (if (string/starts-with text "\\begin{NOTES}") ""))
   ((eq backend 'rst)
    (if (string/starts-with text ".. NOTES::") ""))
   ((eq backend 'html)
    (if (string/starts-with text "<div class=\"NOTES\">") ""))
   ((eq backend 'beamer)
    (let ((text (replace-regexp-in-string "\\\\begin{NOTES}" "\\\\note{" text)))
      (replace-regexp-in-string "\\\\end{NOTES}" "}" text)))
   ))

(eval-after-load 'ox '(add-to-list
                       'org-export-filter-special-block-functions
                       'my/process-NOTES-blocks))
```

5 Handouts: Features for LaTeX

The features introduced in this section are readily available, as the necessary items in the LaTeX preamble enabling the features are already specified in the properties of the `Lectures` subtree in the semester Org files of the Org-Coursepack.

5.1 Inserting Boxed Paragraphs

With the `mdframed` block, users can easily create boxed paragraphs in LaTeX handouts. The example below shows the code for the box and what the box will look like in the handout. Note that the title of the box is written in bold instead of using `#+ATTR_LATEX: :options [frametitle={Title of the box}]` option, so the title gets printed in both reveal.js and LaTeX outputs.

```
#+BEGIN_mdframed
*Title of the box*

Content of the box
#+END_mdframed
```

Title of the box
Content of the box

Optionally, users can choose to add the following code to their init file so `mdframed` boxes are automatically converted to `note` directives in reStructuredText export.

```
(defun my/process-mdframed-blocks (text backend info)
  "Filter mdframed special blocks in export."
  (cond
   ((org-export-derived-backend-p backend 'rst)
      (replace-regexp-in-string ".. mdframed::" ".. note::" text t t))
   ))

(eval-after-load 'ox '(add-to-list
                       'org-export-filter-special-block-functions
                       'my/process-mdframed-blocks))
```

5.2 Organizing Content in Multiple Columns

One can easily make parts of the handout multi-column. The example below shows the code for creating two columns and what that will look like in the handout.

```
  #+LATEX: \begin{multicols}{2}
  This is content in the first column.
```

```
This is content in the first column.
This is content in the first column.

This is content in the second column.
This is content in the second column.
This is content in the second column.
#+LATEX: \end{multicols}
```

This is content in the first column. This is content in the first column. This is content in the first column.

This is content in the second column. This is content in the second column. This is content in the second column.