Premark Documentation

Release 0.1.3

Ethan Swan

Jun 02, 2022

CONTENTS

Premark generates single-file HTML presentations from one or many markdown files, using Remark.js. Based on Remarker by @tylerdave.

License: MIT

Documentation: Read The Docs

CHAPTER

ONE

NOTABLE FEATURES

- Create slides from simple Markdown. Use three dashes on their own line (`---`) to indicate the transition from one slide to another. All other markdown features work as expected.
- The output is always a *single* HTML file. This means you can open it in your browser without spinning up a web server.
  - In contrast, with vanilla Remark, if your main HTML file needs to load any other files then it can't be opened locally without a web server.
- Your slides can be stored in multiple markdown files and Premark will automatically "stitch" them together into a single presentation, even creating title slides for each section if you want.

CHAPTER

TWO

USAGE EXAMPLE

Generate `presentation.html` from Markdown in `slides.md`:

```
premark -o presentation.html slides.md
```

You can also pass in a custom CSS file to style your presentation.

```
premark -o presentation.html --stylesheet style.css slides.md
```

2.1 Creating a Presentation from Multiple Sections

You may wish to build your presentation from multiple markdown files, each representing a "section" of the final slideshow. Premark supports this through the use of a configuration file, where you can specify the order of your sections.

Say you have a folder `md_slides` containing individual sections `intro.md`, `section_1.md`, `section_2.md`, and `closing.md`. You could create a config file like the following:

```
# config.yaml
sections:
- intro.md
- section_1.md
- section_2.md
- closing.md
```

And then invoke Premark with

```
premark -c config.yaml -o slides.html md_slides
```

The resulting `slides.html` will contain all four sections.

CHAPTER

THREE

DOCUMENTATION SECTIONS

3.1 Installation

Note: If you plan to use Premark from the command line, installing Premark using pipx is recommended. This will make the `premark` executable available in all your Python environments, and the package's dependencies will be installed in an isolated environment, meaning they won't affect other work you do in Python.

3.1.1 Installation via pipx

First, install pipx using these instructions.

Then, install Premark using it:

```
pipx install premark
```

3.1.2 Installation via pip (or other package manager)

If you prefer install Premark in a single Python environment, you can just use pip:

```
pip install premark
```

3.2 Using from the Command Line

Commands are generally of the form:

```
premark [OPTIONS] SLIDE_SOURCE
```

The simple invocation below reads input from `slides.md`, parses it, and writes the resulting RevealJS slideshow to `presentation.yaml`.

```
premark -o presentation.html slides.md
```

`SLIDE_SOURCE` can be the name of a single markdown file (formatted as slides, as RevealJS expects) or the name of a folder with multiple such files inside along with a `sections.yaml` file.

Available options are below:

Option	Description
`--version`	Show the version and exit.
`-v, --verbose`	Output debugging info.
`-t, --title TEXT`	HTML title of the presentation.
`-o, --output-file FILENAME`	Write the output to a file instead of STDOUT.
`-m, --metafile TEXT`	File definition for the order of section stitching. Only needed if using a sections folder.
`-c, --css-file PATH`	Custom CSS to be included inline.
`--html-template PATH`	Jinja2 template file for the presentation.
`--help`	Show this message and exit.

3.2.1 Usage Examples

Breaking Markdown into Sections

Premark can combine multiple markdown files containing slide definitions into a single final presentation. Simply put all the markdown files in a single folder (say, `slide_sections`) along with a `sections.yaml` file like the below:

```
sections:
- file: intro.md
- file: agenda.md
- file: closing.md
```

Then, when invoked with the below command, Premark will search that `slide_sections` directory for `intro.md`, `agenda.md`, and `closing.md`, merging them and producing a single presentation from the result.

```
premark -o presentation.html slide_sections
```

Premark can even add title slides to each section. Simply provide a `title` key in the section metadata file:

```
sections:
- file: intro.md
- file: agenda.md
  title: Agenda
- file: closing.md
  title: Final Thoughts
```

Sections without a `title` key will not have a title slide added and aren't counted when numbering the sections.

Custom CSS or HTML

Premark allows you to specify your own CSS or HTML template to be used in the final presentation, through the `--css-file` and `--html-template` options.

```
premark -o presentation.html --css-file styles.css slides.md
```

```
premark -o presentation.html --html-template template.html slides.md
```

3.3 Using as a Python Library

Premark exposes a `Presentation` class that can be used to create presentations from within Python.

```
from premark import Presentation

my_markdown = '''
class: center, middle
# My Presentation

...
'''

# Create a presentation object from some markdown
p = Presentation(markdown=my_markdown)

# Render the presentation as HTML
html = p.to_html()
# You probably want to save the HTML to a file
with open('prez.html', 'w') as f:
    f.write(html)
```

3.3.1 Creating Presentations

There are three ways of creating a new presentation:

1. Passing a markdown source file.
2. Passing a directory of markdown source files.
3. Passing literal markdown text.

1. From a single markdown file

```
p = Presentation('path/to/markdown.md')
```

This is the simplest approach. The argument may be a string, a `pathlib.Path`, or a file-like object (one supporting a `read()` method).

2. From a directory of markdown files

```
p = Presentation('path/to/markdown_directory', config_file='conf.yaml')
```

When passing a directory as the source, a configuration file (in YAML) is required. In this file you must specify the order in which to combine your markdown files. For example:

```
sections:
- intro.md
- section_1.md
- section_2.md
- conclusion.md
```

All of these files must exist inside the source directory.

A more verbose syntax in the `sections` configuration can unlock more Premark features. By specifying sections using both `file` and `title`, Premark will automatically include a title slide before each new section.

```
sections:
- file: intro.md
  title: Introduction
- file: section_1.md
  title: The Interface of My Project
- file: section_2.md
  title: The Implementation of My Project
- file: conclusion.md
  title: Wrapping Up
```

3. From literal markdown

Premark can accommodate markdown already stored in a string. Pass it using the `markdown` keyword argument.

```
md = '# Welcome\n---## Agenda\n1. Content'
p = Presentation(markdown=md)
```

3.3.2 Customizing Presentations

As seen above, literal markdown or a file source is required when creating a new presentation. Other parameters are optional, but can be very useful for customizing your rendered slides.

Parameters:

- `html_template` (str, Path, or file-like) – A file containing a custom HTML Jinja template into which the title, markdown, stylesheet, and remark arguments should be inserted, overriding the default one. (All those fields are expected to be present in the template, e.g. `{{ title }}`)
- `stylesheet` (str, Path, or file-like) – A file containing the CSS styles to insert into the presentation, overriding the default one.
- `title` (str) – The title of the rendered presentation. Has no impact on the slides themslves but is inserted in the HTML title tag.
- `remark_args` (dict) – the arguments to pass to `remark.create`, overriding the defaults.

For full documentation of the available arguments when creating `Presentation`s, see the API docs.

3.3.3 Config Files

While – except for section ordering and titling – all presentation customization can be done without a separate config file, using config files is well worth it when working on presentations that you intend to rebuild repeatedly. In such cases, leaving a `config.yaml` in the directory allows you to version control your Premark configuration.

If the `config_file` argument is passed to `Presentation`, that file is read and its values used as configuration *unless* the same value was also passed as an argument. (Explicit arguments always take priority.) Additionally, the ordering of multiple presentation "sections" can *only* be specified via a config file.

Your config file might look like this:

```
sections:
- intro.md
- agenda.md
- main_content.md
- closing.md
title: The Best Slideshow
stylesheet: assets/styles.css
```

Along with (or instead of) `stylesheet` and `title`, `html_template` is also an accepted argument.

3.3.4 Laying Out Your Project

In most cases, if you're using Premark, you have one or several markdown files containing slides and those live in a project folder of some sort – and that project is in version control. A good layout in such cases is to keep your config file in the base of the repo and name it `premark.yaml`, and your slides in an `premark_slides` folder. If you have custom HTML or CSS files, put them in an `premark_assets/` folder.

```
myproject
├── premark.yaml
├── premark_assets
|   ├── template.html
|   └── styles.css
└── premark_slides
    ├── agenda.md
    ├── closing.md
    ├── exercises.md
    ├── intro.md
    └── main_content.md
```

And the contents of `premark.yaml` might look like this:

```
sections:
- file: intro.md
- file: agenda.md
- file: main_content.md
- file: closing.md
- file: exercises.md
stylesheet: premark_assets/styles.css
html_template: premark_assets/template.html
title: A Wild Ride
```

With this format, you can store Premark presentations in the same folder as related projects. This may not be necessary in most cases, but this kind of organization is very handy occasionally.

3.3.5 Exporting Presentations (i.e. *Rendering*)

There's only one format for exporting presentations: HTML. The `.to_html` method performs the conversion for you:

```
p = Presentation(...)
html = p.to_html()
```

That's it!

API docs

3.4 API Documentation

class premark.presentation.**Presentation**(*source: Optional[Union[str, pathlib.Path, premark.utils.Readable]] = None*, *markdown: Optional[str] = None*, *remark_args: Optional[dict[str, Union[str, bool]]] = None*, *html_template: Optional[Union[str, pathlib.Path, premark.utils.Readable]] = None*, *stylesheet: Optional[Union[str, pathlib.Path, premark.utils.Readable]] = None*, *title: Optional[str] = None*, *config_file: Optional[Union[str, pathlib.Path, premark.utils.Readable]] = None*)

A RemarkJS presentation.

classmethod from_presentations(*presentations: Iterable[*premark.presentation.Presentation*]*) → *premark.presentation.Presentation*

Create a single presentations by merging others together.

Parameters **presentations** – An iterable of Presentation objects

Returns The resulting, merged presentation

Return type *Presentation*

to_html() → str

Convert the presentation to HTML.

Returns An HTML rendering of the presentation.

Return type str

3.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.5.1 Types of Contributions

Report Bugs

Report bugs on the Issues Page.

If you are reporting a bug, please include:

- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Premark can always use more documentation, whether as part of the official docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is via the Issues Page.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.5.2 Get Started!

Ready to contribute? Here's how to set up Premark for local development.

1. Fork the Premark repo on GitHub.
2. Clone your fork locally:

```
$ git clone https://github.com/your_name_here/premark.git
```

3. Install your local copy into a venv. You'll Python 3.9 or above for this step. Once you've cded into the base folder of the repository, follow these steps to get a venv set up for development on Premark:

```
$ python3 -m venv .venv

# Necessary for editable installs of a pyproject.toml codebase.
$ pip install --upgrade pip

$ . .venv/bin/activate

$ pip install -e ".[dev]"
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. Make your changes (update code, add tests, change docs, etc.).

6. When you're done making changes, check that your changes pass flake8, mypy, and the tests, including testing all supported Python versions with via tox:

```
$ flake8 premark tests
$ mypy premark
$ pytest
$ tox
```

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request (to the develop branch of the original Premark repo) through the GitHub website.

3.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The PR should include new or updated tess for any functionality that has been changed.
2. New functions and classes should have descriptive docstrings adhering to the style of the rest of the package.
3. The CHANGELOG.md file should be updated with an account of your changes.
4. The code should work for Python versions 3.9 and above. Check and make sure that the tests pass for all supported Python versions.

3.5.4 Releasing a New Version

The GitHub Actions pipeline is set up such that any Git tag starting with "v" will be built and published as a new version. Ideally, merge changes into `main` and tag that commit (`git tag -a v1.2.3`), then push to GitHub including tags (`git push origin main --tags`).

3.6 Authors

Premark is maintained by Ethan Swan, @eswan18 on GitHub.

However, much credit is due to Dave Forgac (@tylerdave), for the creation of Remarker, from which Premark is forked.

3.6.1 Other Contributors

None yet. Why not be the first? See the *contributing page*.

3.7 Changelog

3.7.1 Version 0.1.3

- Add CSS styles for flexboxes, embeddable in markdown with a syntax like:

```
.flex[
.half-flex-container[
... my first column ...
]
.half-flex-container[
... my other column
]
]
```

- Add a `.allow-wrap` CSS class for forcing code to wrap in markdown
- Update html template so that autogenerated section title slides have the `premark-section-title` class, which makes their headings larger via CSS.

3.7.2 Version 0.1.2

- Update manifest file to include Premark's default configuration when the package is installed.

3.7.3 Version 0.1.1

- Fix section auto-numbering (see #10).

3.7.4 Version 0.1.0

- Overhaul interface of both the CLI and the `Presentation` class.
- Support config files as yaml.

3.7.5 Version 0.0.9

- First release

INDEX

F

P

T