lookatme

Release v0.2.0

Dec 04, 2019

Contents

`lookatme` is an interactive, terminal-based markdown presentation tool that supports:

- Themes
- Syntax highlighting
- Styling and settings embedded within the Markdown YAML header
- Embedded terminals as part of a presentation
- Live and manual source reloading
- User-contrib behavior overrides and extensions

CHAPTER 1

Tour

CHAPTER 2

TL;DR Getting Started

Install `lookatme` with:

```
pip install lookatme
```

Run lookatme on slides written in Markdown:

```
lookatme slides.md
```

Slides are separated with `---` hrules:

```
# Slide 1

Some text

---

# Slide 2

More text
```

A basic, optional YAML header may be included at the top of the slides:

```
---
title: Slides Presentation
author: Me Not You
date: 2019-12-02
---

# Slide 1

Some text
```

2.1 Getting Started

2.1.1 Installation

`lookatme` can be installed with pip using the command:

```
pip install lookatme
```

2.1.2 Usage

The `lookatme` CLI has a few options to control it's behavior:

```
Usage: lookatme [OPTIONS] [INPUT_FILE]

  lookatme - An interactive, terminal-based markdown presentation tool.

Options:
  --debug
  -l, --log PATH
  -t, --theme [dark|light]
  -s, --style␣
↪[default|emacs|friendly|colorful|autumn|murphy|manni|monokai|perldoc|pastie|borland|trac|native|fru
↪light|paraiso-dark|lovelace|algol|algol_nu|arduino|rainbow_dash|abap|solarized-
↪dark|solarized-light|sas|stata|stata-light|stata-dark]
  --dump-styles                   Dump the resolved styles that will be used
                                  with the presentation to stdout
  --live, --live-reload           Watch the input filename for modifications
                                  and automatically reload
  --help                          Show this message and exit.
```

`--live` / `--live-reload`

This flag turns on live reloading within lookatme. If the input markdown is a filepath (and not stdin), the filepath with be watched for changes to its modification time. If a change to the file's modification time is observed, the slide deck is re-read and rendered, keeping the current slide in focus.

If your editor supports saving with every keystroke, instant slide updates are possible:

`--debug` and `--log`

Turns on debug logging for lookatme. The debug log will be created in your platform's temporary directory by default and will be named `lookatme.log`:

```
$> lookatme slides.md --debug

# in another terminal
$> tail -f /tmp/lookatme.log
DEBUG:lookatme.RENDER:  Rendering token {'type': 'heading', 'level': 2, 'text': 'TOC'}
DEBUG:lookatme.RENDER:  Rendering token {'type': 'list_start', 'ordered': False}
DEBUG:lookatme.RENDER:    Rendering token {'type': 'list_item_start'}
DEBUG:lookatme.RENDER:      Rendering token {'type': 'text', 'text': '[Features](
↪#features)'}
```

(continues on next page)

(continued from previous page)

```
DEBUG:lookatme.RENDER:      Rendering token {'type': 'list_start', 'ordered': False}
DEBUG:lookatme.RENDER:        Rendering token {'type': 'list_item_start'}
```

You may set a custom log location with the `--log` flag

`--theme`

Themes in lookatme are pre-defined stylings. Lookatme comes with two built-in themes: `dark` and `light`. These themes are intended to look good on dark terminals and light terminals.

See the *Dark Theme* and *Light Theme* pages for more details. See the *Style Precedence* page for details on the order style overrides and settings are applied.

`--style`

This option overrides the Pygments syntax highlighting style to use. See the *Style Precedence* for details about style overriding order.

At the time of this writing, available Pygments style options include:

- default
- emacs
- friendly
- colorful
- autumn
- murphy
- manni
- monokai
- perldoc
- pastie
- borland
- trac
- native
- fruity
- bw
- vim
- vs
- tango
- rrt
- xcode
- igor
- paraiso-light

- paraiso-dark
- lovelace
- algol
- algol_nu
- arduino
- rainbow_dash
- abap
- solarized-dark
- solarized-light
- sas
- stata
- stata-light
- stata-dark

`--dump-styles`

Print the final, resolved style definition that will be used to render the markdown as currently specified on the command-line. See the *Style Precedence* section for details on how this works.

E.g.:

```
lookatme examples/tour.md -theme --style solarized-dark --dump-styles
```

2.2 Slides

Slides in `lookatme` are:

- Separated by hrule elements: `---` in Markdown
- Resized to fit the current window

2.2.1 Metadata

Slide metadata is contained within an optional YAML header:

```
---
title: TITLE
author: AUTHOR
date: 2019-12-02
extensions: []
styles: {}
---
```

Extensions

Extensions are lookatme contrib modules that redefine lookatme behavior. E.g., the `lookatmecontrib.calendar` example in the examples folder redefines the `render_code` function found in `lookatme/render/markdown_block.py`.

The original `render_code` function gives contrib extensions first-chance at handling any function calls. Contrib extensions are able to ignore function calls, and thus allow the default `lookatme` behavior, by raising the *`IgnoredByContrib`* exception:

```
import datetime
import calendar
import urwid


from lookatme.exceptions import IgnoredByContrib


def render_code(token, body, stack, loop):
    lang = token["lang"] or ""
    if lang != "calendar":
        raise IgnoredByContrib()

    today = datetime.datetime.utcnow()
    return urwid.Text(calendar.month(today.year, today.month))
```

Styles

In addition to the `--style` and `--theme` CLI options for lookatme, the slide metadata may explicitly override styling behaviors within lookatme:

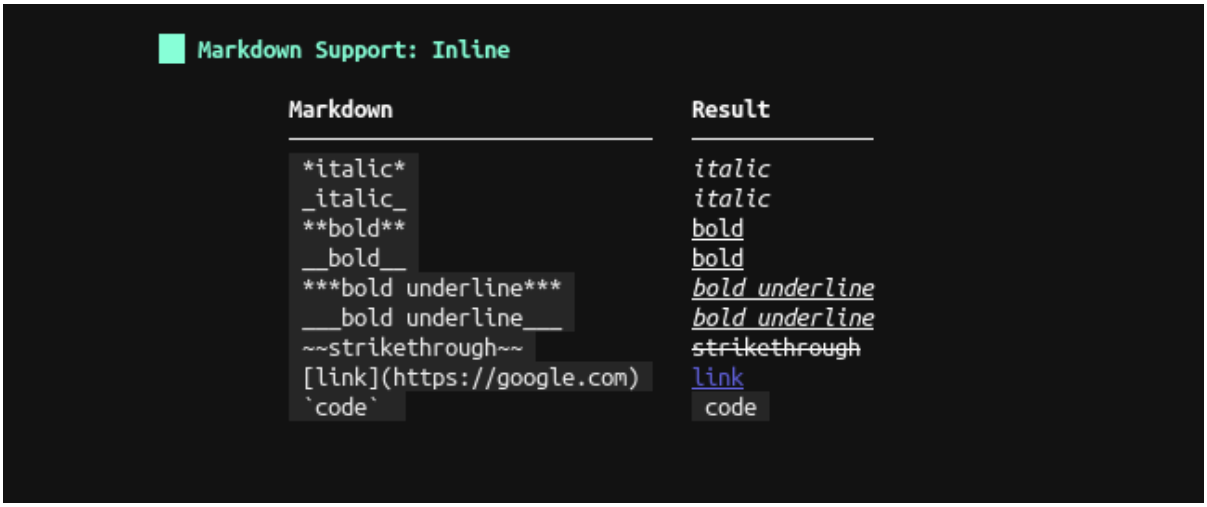
```
---
title: TITLE
author: AUTHOR
date: 2019-12-02
styles:
  style: monokai
  table:
    column_spacing: 3
    header_divider: "-"
---

# Slide 1

text
```

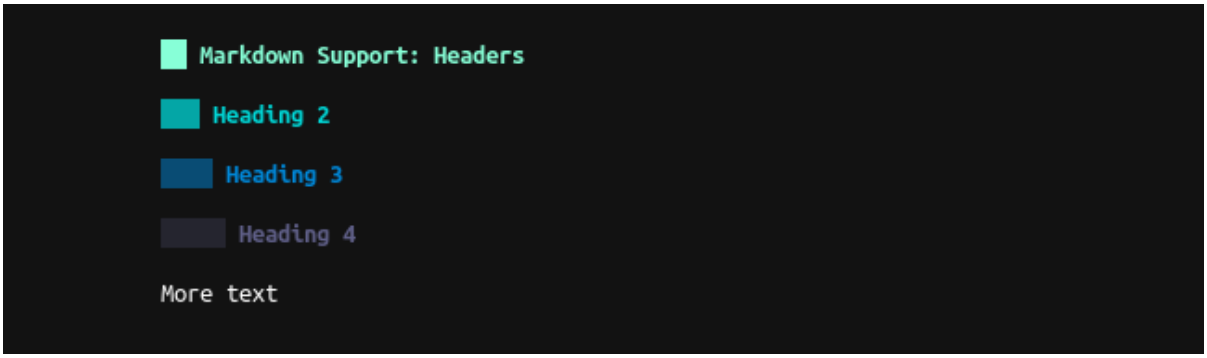
The final, resolved styling settings that will be used when displaying a markdown source is viewable by adding the `--dump-styles` flag as a command-line argument.

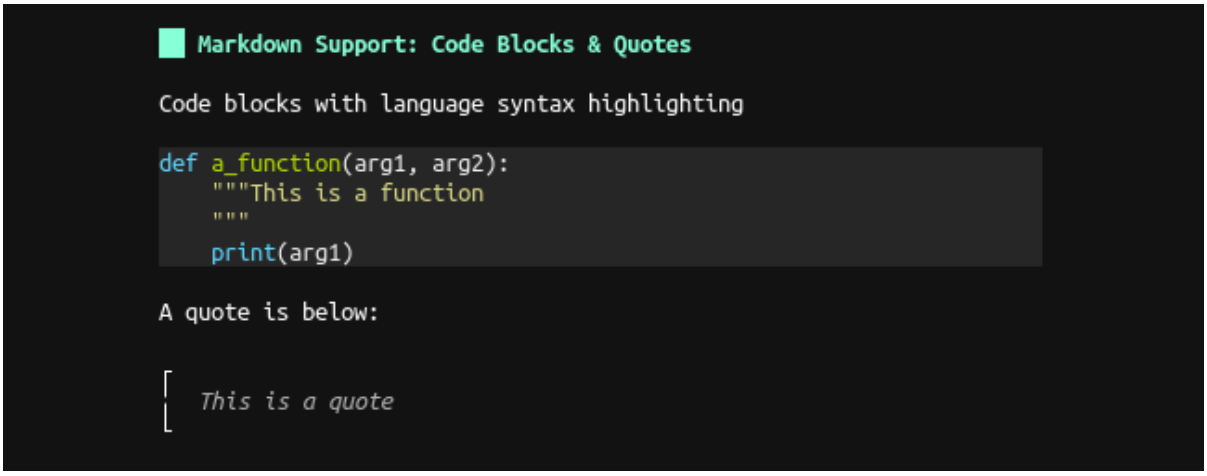
See the *Default Style Settings* for a full list of available, overrideable styles.

2.3 Dark Theme

The dark theme is intended to appear well on terminals with dark backgrounds

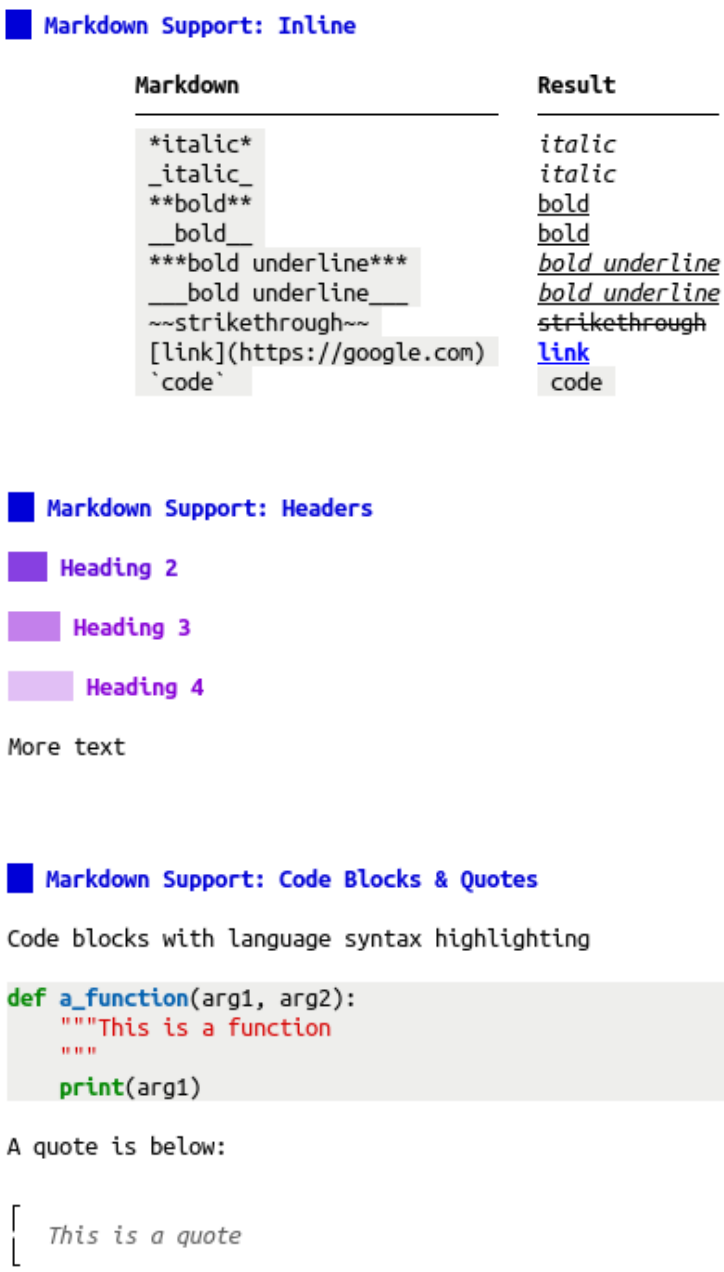






2.4 Light Theme

The light theme is intended to appear well on terminals with light backgrounds



2.5 Style Precedence

Styling may be set in three locations in lookatme:

1. In a theme
2. In a slide's YAML header
3. On the command-line

When constructing the final, resolved style set that will be used to render markdown, lookatme starts with the default style settings defined in `lookatme.schemas`, and then applies overrides in the order specified above.

Overrides are applied by performing a deep merge of nested dictionaries. For example, if the default styles defined in schemas.py were:

```
headings:
  "1":
    fg: "#33c,bold"
    bg: "default"
  "2":
    fg: "#222,bold"
    bg: "default"
```

... and if the style overrides defined by a theme were:

```
headings:
  "1":
    bg: "#f00"
```

... and if the style overrides defined in the slide YAML header were:

```
headings:
  "2":
    fg: "#f00,bold,underline"
```

The final, resolved style settings for rendering the markdown would be:

```
headings:
  "1":
    fg: "#33c,bold"
    bg: "#f00" # from the theme
  "2":
    fg: "#f00,bold,underline" # from the slide YAML header
    bg: "default"
```

2.5.1 Default Style Settings

The default styles and formats are defined in the marshmallow schemas in *lookatme.schemas*. The dark theme is an empty theme with no overrides (the defaults *are* the dark theme):

```
bullets:
  '1': "•"
  '2': ""
  '3': "○"
  default: "•"
headings:
  '1':
    bg: default
    fg: '#9fc,bold'
    prefix: " "
    suffix: ""
  '2':
    bg: default
    fg: '#1cc,bold'
    prefix: " "
    suffix: ""
  '3':
    bg: default
```

(continues on next page)

(continued from previous page)

```
    fg: '#29c,bold'
    prefix: " "
    suffix: ""
  '4':
    bg: default
    fg: '#66a,bold'
    prefix: " "
    suffix: ""
  default:
    bg: default
    fg: '#579,bold'
    prefix: " "
    suffix: ""
link:
  bg: default
  fg: '#228,underline'
quote:
  top_corner: ""
  bottom_corner: "└"
  side: ""
  style:
    bg: default
    fg: italics,#aaa
style: solarized-dark
table:
  column_spacing: 3
  header_divider: "─"
```

2.6 lookatme

2.6.1 lookatme package

Subpackages

lookatme.contrib package

Submodules

lookatme.contrib.terminal module

This module defines a built-in contrib module that enables terminal embedding within a slide.

lookatme.contrib.terminal.**render_code**(*token*, *body*, *stack*, *loop*)

lookatme.contrib.terminal.**shutdown**()

Module contents

This module handles loading and using lookatme_contriba modules

Contrib modules are directly used

lookatme.contrib.**contrib_first**(*fn*)
A decorator that allows contrib modules to override default behavior of lookatme. E.g., a contrib module may override how a table is displayed to enable sorting, or enable displaying images rendered with ANSII color codes and box drawing characters, etc.

Contrib modules may ignore chances to override default behavior by raising the `lookatme.contrib.IgnoredByContrib` exception.

lookatme.contrib.**load_contribs**(*contrib_names*)
Load all contrib modules specified by `contrib_names`. These should all be namespaced packages under the `lookatmecontrib` namespace. E.g. `lookatmecontrib.calendar` would be an extension provided by a contrib module, and would be added to an `extensions` list in a slide's YAML header as `calendar`.

lookatme.contrib.**shutdown_contribs**()
Call the shutdown function on all contrib modules

lookatme.render package

Submodules

lookatme.render.asciinema module

lookatme.render.markdown_block module

Defines render functions that render lexed markdown block tokens into urwid representations

lookatme.render.markdown_block.**render_block_quote_end**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_block_quote_start**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_code**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_heading**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_list_end**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_list_item_end**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_list_item_start**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_list_start**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_loose_item_start**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_paragraph**(*token*, *body*, *stack*, *loop*)

lookatme.render.markdown_block.**render_table**(*token*, *body*, *stack*, *loop*)
Render a table token

lookatme.render.markdown_block.**render_text**(*token=None*, *body=None*, *stack=None*, *loop=None*, *text=None*)

lookatme.render.markdown_inline module

Defines render functions that work with mistune's markdown inline lexer render interface

lookatme.render.markdown_inline.**autolink**(*link_uri*, *is_email=False*)

lookatme.render.markdown_inline.**codespan**(*text*, *old_styles*)

lookatme.render.markdown_inline.**double_emphasis**(*text*, *old_styles*)

lookatme.render.markdown_inline.**emphasis**(*text*, *old_styles*)

lookatme.render.markdown_inline.**escape**(*text*)
Render inline markdown text with no changes

lookatme.render.markdown_inline.**expanded_styles**(*fn*)

lookatme.render.markdown_inline.**footnote_ref**(*key*, *index*)

lookatme.render.markdown_inline.**image**(*link_uri*, *title*, *text*)

lookatme.render.markdown_inline.**inline_html**(*html*)

lookatme.render.markdown_inline.**linebreak**()

lookatme.render.markdown_inline.**link**(*link_uri*, *title*, *link_text*)

lookatme.render.markdown_inline.**placeholder**()
The starting point of the rendering. The final result will be this returned list with all inline markdown tokens translated into urwid objects

lookatme.render.markdown_inline.**render_no_change**(*text*)
Render inline markdown text with no changes

lookatme.render.markdown_inline.**strikethrough**(*text*, *old_styles*)

lookatme.render.markdown_inline.**text**(*text*)
Render inline markdown text with no changes

lookatme.render.markdown_inline.**underline**(*text*, *old_styles*)

lookatme.render.pygments module

Pygments related rendering

class lookatme.render.pygments.**UrwidFormatter**(***options*)
Bases: pygments.formatter.Formatter

Formatter that returns [(text,attrspec), . . .], where text is a piece of text, and attrspec is an urwid.AttrSpec

classmethod findclosest(*colstr*, *colors=256*)
Takes a hex string and finds the nearest color to it.

Returns a string urwid will recognize.

findclosestattr(*fgcolstr=None*, *bgcolstr=None*, *othersettings=''*, *colors=256*)
Takes two hex colstring (e.g. 'ff00dd') and returns the nearest urwid style.

format(*tokensource*, *outfile*)
Format `tokensource`, an iterable of `(tokentype, tokenstring)` tuples and write it into `outfile`.

formatgenerator(*tokensource*)
Takes a token source, and generates (tokenstring, urwid.AttrSpec) pairs

style

lookatme.render.pygments.**get_formatter**(*style_name*)

lookatme.render.pygments.**get_lexer**(*lang*, *default='text'*)

lookatme.render.pygments.**get_style**(*style_name*)

lookatme.render.pygments.**render_text**(*text*, *lang='text'*, *style_name=None*, *plain=False*)
Render the provided text with the pygments renderer

Module contents

lookatme.themes package

Submodules

lookatme.themes.dark module

Defines styles that should look good on dark backgrounds

lookatme.themes.light module

Module contents

Defines the built-in styles for lookatme

lookatme.themes.**ensure_defaults**(*mod*)
Ensure that all required attributes exist within the provided module

lookatme.widgets package

Submodules

lookatme.widgets.clickable_text module

This module contains code for ClickableText

class lookatme.widgets.clickable_text.**ClickableText**(*markup*, *align='left'*, *wrap='space'*, *layout=None*)
Bases: urwid.widget.Text

Allows clickable/changing text to be part of the Text() contents

mouse_event(*size*, *event*, *button*, *x*, *y*, *focus*)
Handle mouse events!

signals = ['click', 'change']

class lookatme.widgets.clickable_text.**LinkIndicatorSpec**(*link_label*, *link_target*, *orig_spec*)
Bases: urwid.display_common.AttrSpec

Used to track a link within an urwid.Text instance

lookatme.widgets.table module

Defines a basic Table widget for urwid

class lookatme.widgets.table.**Table**(*rows*, *headers=None*, *aligns=None*)
Bases: urwid.container.Pile

Create a table from a list of headers, alignment values, and rows.

calc_column_maxes()

create_cells(*body_rows*, *modifier=None*)
Create the rows for the body, optionally calling a modifier function on each created cell Text. The modifier must accept an urwid.Text object and must return an urwid.Text object.

render(*size*, *focus=False*)
Do whatever needs to be done to render the table

set_column_maxes()
Calculate and set the column maxes for this table

signals = ['change']

watch(*w*)
Watch the provided widget w for changes

Module contents

Submodules

lookatme.config module

Config module for lookatme

lookatme.exceptions module

Exceptions used within lookatme

exception lookatme.exceptions.**IgnoredByContrib**
Bases: Exception

Raised when a contrib module's function chooses to ignore the function call.

lookatme.log module

Logging module

lookatme.log.**create_log**(*log_path*)
Create a new log that writes to log_path

lookatme.log.**create_null_log**()
Create a logging object that does nothing

lookatme.parser module

This module defines the parser for the markdown presentation file

class lookatme.parser.**Parser**
Bases: object

A parser for markdown presentation files

parse(*input_data*)
Parse the provided input data into a Presentation object

Parameters **input_data** (*str*) – The input markdown presentation to parse

Returns Presentation

parse_meta(*input_data*)
Parse the PresentationMeta out of the input data

Parameters **input_data** (*str*) – The input data string

Returns tuple of (remaining_data, meta)

parse_slides(*input_data*)
Parse the Slide out of the input data

Parameters **input_data** (*str*) – The input data string

Returns tuple of (remaining_data, slide)

lookatme.pres module

Defines Presentation specific objects

class lookatme.pres.**Presentation**(*input_stream*, *theme*, *style_override=None*, *live_reload=False*)
Bases: object

Defines a presentation

reload(*data=None*)
Reload this presentation

Parameters **data** (*str*) – The data to render for this slide deck (optional)

reload_watcher()
Watch for changes to the input filename, automatically reloading when the modified time has changed.

run(*start_slide=0*)
Run the presentation!

lookatme.schemas module

Defines all schemas used in lookatme

class lookatme.schemas.**BlockQuoteSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)
Bases: marshmallow.schema.Schema

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**BulletsSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)

Bases: marshmallow.schema.Schema

class Meta

Bases: object

include = {'1': <fields.String(default='•', attribute=None, validate=None, require

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**HeadingStyleSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)

Bases: marshmallow.schema.Schema

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**HeadingsSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)

Bases: marshmallow.schema.Schema

class Meta

Bases: object

include = {'1': <fields.Nested(default={'fg': '#9fc,bold', 'bg': 'default', 'pre

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**MetaSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)

Bases: marshmallow.schema.Schema

The schema for presentation metadata

class Meta

Bases: object

render_module

alias of *YamlRender*

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**NoDatesSafeLoader**(*stream*)
Bases: yaml.loader.SafeLoader

classmethod remove_implicit_resolver(*tag_to_remove*)
Remove implicit resolvers for a particular tag

Takes care not to modify resolvers in super classes.

We want to load datetimes as strings, not dates, because we go on to serialise as json which doesn't have the advanced types of yaml, and leads to incompatibilities down the track.

yaml_implicit_resolvers = {'': [('tag:yaml.org,2002:null', re.compile('^(?: ~\n |nul

class lookatme.schemas.**StyleFieldSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)
Bases: marshmallow.schema.Schema

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**StyleSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)
Bases: marshmallow.schema.Schema

Styles schema for themes and style overrides within presentations

class Meta
Bases: object

render_module
alias of *YamlRender*

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**TableSchema**(**, only: Union[Sequence[str], Set[str]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Dict[KT, VT] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: str = None*)
Bases: marshmallow.schema.Schema

opts = <marshmallow.schema.SchemaOpts object>

class lookatme.schemas.**YamlRender**
Bases: object

dumps()

loads()

lookatme.slide module

Slide info holder

class lookatme.slide.**Slide**(*tokens*, *md=None*, *number=0*)
Bases: object

This class defines a single slide. It operates on mistune's lexed tokens from the input markdown

lookatme.tui module

This module defines the text user interface (TUI) for lookatme

class lookatme.tui.**MarkdownTui**(*pres*, *palette*, *start_idx=0*)
Bases: urwid.container.Frame

keypress(*size*, *key*)
Handle keypress events

prep_pres(*pres*, *start_idx=0*)
Prepare the presentation for displaying/use

reload()
Reload the input, keeping the current slide in focus

run()

update()

update_body()
Render the provided slide body

update_creation()
Update the author and date

update_slide_num()
Update the slide number

update_title()
Update the title

class lookatme.tui.**SlideRenderer**(*loop*)
Bases: threading.Thread

daemon = True

flush_cache()
Clea everything out of the queue and the cache.

get_slide(*slide_number*)
Fetch the slide from the cache

queue_render(*slide*)
Queue up a slide to be rendered.

render_slide(*slide*, *force=False*)
Render a slide, blocking until the slide completes. If force is True, rerender the slide even if it is in the cache.

run()
Run the main render thread

stop()

lookatme.tui.**create_tui**(*pres*, *start_slide=0*)
Run the provided presentation

Parameters **start_slide** (*int*) – 0-based slide index

lookatme.tui.**text**(*style*, *data*, *align='left'*)

lookatme.utils module

lookatme.utils.**dict_deep_update**(*to_update*, *new_vals*)
Deeply update the to_update dict with the new_vals

lookatme.utils.**flatten_text**(*text*, *new_spec=None*)
Return a flattend list of tuples that can be used as the first argument to a new urwid.Text().

Parameters

- **text** (*urwid.Text*) – The text to flatten
- **new_spec** (*urwid.AttrSpec*) – A new spec to merge with existing styles

Returns list of tuples

lookatme.utils.**get_fg_bg_styles**(*style*)

lookatme.utils.**overwrite_spec**(*orig_spec*, *new_spec*)

lookatme.utils.**pile_add**(*pile*, *widgets*)

lookatme.utils.**resolve_bag_of_text_markup_or_widgets**(*items*)
Resolve the list of items into either contiguous urwid.Text() instances, or pre-existing urwid.Widget objects

lookatme.utils.**row_text**(*rendered_row*)
Return all text joined together from the rendered row

lookatme.utils.**spec_from_style**(*styles*)
Create an urwid.AttrSpec from a {fg:"", bg:""} style dict. If styles is a string, it will be used as the foreground

lookatme.utils.**styled_text**(*text*, *new_styles*, *old_styles=None*, *supplement_style=False*)
Return a styled text tuple that can be used within urwid.Text.

Note: If an urwid.Text instance is passed in as the `text` parameter, alignment values will be lost and must be explicitly re-added by the caller.

---

lookatme.utils.**translate_color**(*raw_text*)

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

K

L

M

N

O

P

Q

R

S

T

U

W

Y