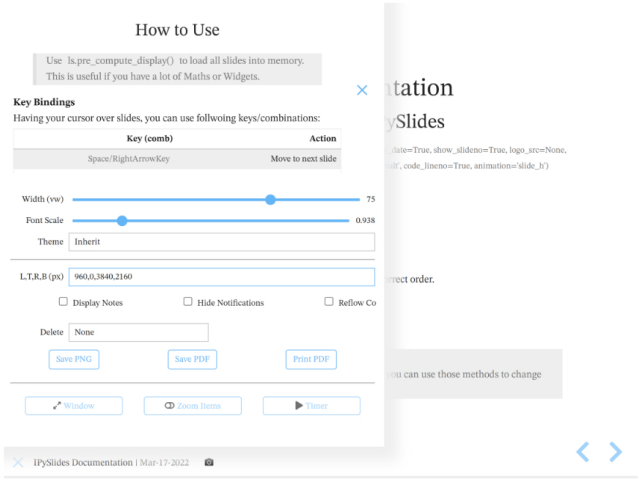

tation

ySlides

_date=True, show_slideno=True, logo_src=None,

ult', code_lineno=True, animation='slide_h')

Width (vw) 75

Font Scale 0.938

Theme Light

L,T,R,B (px) 960,0,3840,2160

Display Notes Hide Notifications Reflow Co

rrect order.

Delete None

Save PNG Save PDF Print PDF

you can use those methods to change

Window Zoom Items Timer

IPySlides Documentation

Creating slides with IPySlides

LiveSlides(center=True, content_width='90%', footer_text='IPySlides | github-link', show_date=True, show_slideno=True, logo_src=None, font_scale=1, text_font='sans-serif', code_font='var(--jp-code-font-family)', code_style='default', code_lineno=True, animation='slide_h')

Interactive Slides in IPython Notebook. Only one instance can exist.

Example

```
import ipyslides as isd
ls = isd.LiveSlides()
ls.demo() # Load demo slides
ls.from_markdown(...) # Load slides from markdown files
```

Instead of builtin `print` in slides use following to display printed content in correct order.

```
with ls.print_context():
  print('something')
  function_that_prints_something()
```

> ls.demo and ls.from_markdown overwrite all previous slides.
>
> All arguments are passed to corresponding methods in ls.settings , so you can use those methods to change settings as well.

Adding Slides

Besides functions below, you can add slides with `%%title` and `%%slide <slide number>` magics as well.

LiveSlides.**title**(**css_props)

Use this context manager to write title.

`css_props` are applied to current slide. `-` -> `_` as `font-size` -> `font_size` in python.

LiveSlides.**slide**(slide_number, **css_props)

Use this context manager to generate any number of slides from a cell

`css_props` are applied to current slide. `-` -> `_` as `font-size` -> `font_size` in python.

LiveSlides.**frames**(slide_number, *objs, repeat=False, frame_height='auto', **css_props)

Decorator for inserting frames on slide, define a function with one argument acting on each obj in objs.

You can also call it as a function, e.g. `.frames(slide_number = 1,1,2,3,4,5)()` becuase required function is `write` by defualt.

```
@slides.frames(1,a,b,c) # slides 1.1, 1.2, 1.3 with content a,b,c
def f(obj):
  do_something(obj)

slides.frames(1,a,b,c)() # Auto writes the frames with same content as above
slides.frames(1,a,b,c, repeat = True)() # content is [a], [a,b], [a,b,c] from top to bottom
slides.frames(1,a,b,c, repeat = [(0,1),(1,2)])() # two frames with content [a,b] and [b,c]
```

Parameters

- slide_number: (int) slide number to insert frames on.
- objs: expanded by * (list, tuple) of objects to write on frames. If repeat is False, only one frame is generated for each obj.
- repeat: (bool, list, tuple) If False, only one frame is generated for each obj.
  If True, one frame are generated in sequence of ojects linke `[a,b,c]` will generate 3 frames with [a], [a,b], [a,b,c] to given in function and will be written top to bottom.
  If list or tuple, it will be used as the sequence of frames to generate and number of frames = len(repeat).
  [(0,1),(1,2)] will generate 2 frames with [a,b] and [b,c] to given in function and will be written top to bottom or the way you write in your function.
- frame_height: ('N%', 'Npx', 'auto') height of the frame that keeps incoming frames object at static place.

No return of defined function required, if any, only should be display/show etc.
`css_props` are applied to all slides from *objs. `-` -> `_` as `font-size` -> `font_size` in python.

Adding Content

Besides functions below, you can add content to slides with `display(obj)` as well.

LiveSlides.**write**(*columns, width_percents=None, className=None)

Writes markdown strings or IPython object with method `_repr_<html,svg,png,...>_` in each column of same with. If width_percents is given, column width is adjusted.
Each column should be a valid object (text/markdown/html/ have *repr or to* method) or list/tuple of objects to form rows or explictly call `rows`.

- Pass int,float,dict,function etc. Pass list/tuple in a wrapped list for correct print as they used for rows writing too.
- Give a code object from `ipyslides.get_cell_code()` to it, syntax highlight is enabled.
- Give a matplotlib `figure/Axes` to it or use `ipyslides.objs_formatter.plt2html()`.
- Give an interactive plotly figure.
- Give a pandas dataframe `df` or `df.to_html()`.
- Give any object which has `to_html` method like Altair chart. (Note that chart will not remain interactive, use display(chart) if need interactivity like brushing etc.)
- Give an IPython object which has `_repr_<repr>_` method where is one of ('html','markdown','svg','png','jpeg','javascript','pdf','pretty','json','latex').
- Give a function/class/module (without calling) and it will be displayed as a pretty printed code block.

If an object is not in above listed things, `obj.__repr__()` will be printed. If you need to show other than **repr**, use `display(obj)` outside `write` command or use
methods specific to that library to show in jupyter notebook.

If you give a className, add CSS of it using `format_css` function and provide it to `write` function.
Get a list of already available classes using `slides.css_styles`. For these you dont need to provide CSS.

Note: Use `keep_format` method to bypass markdown parser, for example `keep_format(altair_chart.to_html())`.
Note: You can give your own type of data provided that it is converted to an HTML string.
Note: `_repr_<format>_` takes precedence to `to_<format>` methods. So in case you need specific output, use `object.to_<format>`.

LiveSlides.**iwrite**(*columns, width_percents=None, className=None)

Each obj in columns could be an IPython widget like `ipywidgets`, `bqplots` etc
or list/tuple (or wrapped in `rows` function) of widgets to display as rows in a column.
Other objects (those in `write` command) will be converted to HTML widgets if possible.
Object containing javascript code may not work, use `write` command for that.

If you give a className, add CSS of it using `format_css` function and provide it to `iwrite` function.
Get a list of already available classes using `slides.css_styles`. For these you dont need to provide CSS.

Returns: writer, columns as reference to use later and update. rows are packed in columns.

Examples:

```
writer, x = iwrite('X')
writer, (x,y) = iwrite('X','Y')
writer, (x,y) = iwrite(['X','Y'])
writer, [(x,y),z] = iwrite(['X','Y'],'Z')
#We unpacked such a way that we can replace objects with new one using `grid.update`
new_obj = writer.update(x, 'First column, first row with new data') #You can update same `new_obj` wi
th it's own widget methods.
```

Adding Speaker Notes

LiveSlides.notes.**insert**(content)

Add notes to current slide. Content could be any object except javascript and interactive widgets.

Displaying Source Code

LiveSlides.source.**context**(collapsed=False, focus_lines=None)

Excute and displays source code in the context manager. Set `collapsed = True` to display in collapse.

`foucs_lines` is a list/tuple/range of line index to be highlighted. Useful when source is written inside context manager itself.

Usage:

```
with source.context() as s: #if not used as `s`, still it is stored in variable `__current_source_code__` that you can acess by this name or from `LiveSlides.current_source`
  do_something()
  #s is the source code that will be avaialble outside the context manager
write(s)
#s.raw, s.value are accesible attributes.
#s.focus_lines, s.show_lines are methods that return object of same type.
# iwite(s) will update the source even inside the context manager.
```

LiveSlides.source.**from_callable**(callable)

Returns source object from a given callable [class,function,module,method etc.] with `show_lines` and `focus_lines` methods.

LiveSlides.source.**from_file**(filename, language='python', name=None)

Returns source object with `show_lines` and `focus_lines` methods. `name` is alternate used name for language

LiveSlides.source.**from_string**(text, language='python', name=None)

Creates source object from string. `name` is alternate used name for language

Layout and Theme Settings

LiveSlides.settings.**code_lineno**(b=True)

Set code line numbers to be shown or not.

LiveSlides.settings.**set_animation**(name)

Set animation style or pass None to disable animation.

LiveSlides.settings.**set_code_style**(style='default', background='var(--secondary-bg)')

Set code style CSS. Use background for better view of your choice.

LiveSlides.settings.**set_font_family**(text_font=None, code_font=None)

Set main fonts for text and code.

LiveSlides.settings.**set_font_scale**(font_scale=1)

Set font scale to increase or decrease text size. 1 is default.

LiveSlides.settings.**set_footer**(text='', show_slideno=True, show_date=True, _number_str='')

Set footer text. `text` should be string. `show_slideno` and `show_date` are booleans.

LiveSlides.settings.**set_layout**(center=True, content_width='100%')

Central aligment of slide by default. If False, left-top aligned.

LiveSlides.settings.**set_logo**(src, width=80, top=0, right=16)

`src` should be PNG/JPEG file name or SVG string. width,top,right are pixels, should be integer.

Useful Functions for Rich Content

LiveSlides.**alert**(text)

Alerts text!

LiveSlides.**block**(title, *objs, bg='olive')

Format a block like in LATEX beamer. *objs expect to be writable with `write` command.

LiveSlides.**block_b**(title, *objs)

See documentation of `block`.

LiveSlides.**block_c**(title, *objs)

See documentation of `block`.

LiveSlides.**block_g**(title, *objs)

See documentation of `block`.

LiveSlides.**block_k**(title, *objs)

See documentation of `block`.

LiveSlides.**block_m**(title, *objs)

See documentation of `block`.

LiveSlides.**block_o**(title, *objs)

See documentation of `block`.

LiveSlides.**block_p**(title, *objs)

See documentation of `block`.

LiveSlides.**block_r**(title, *objs)

See documentation of `block`.

LiveSlides.**block_w**(title, *objs)

See documentation of `block`.

LiveSlides.**block_y**(title, *objs)

See documentation of `block`.

LiveSlides.**bokeh2html**(bokeh_fig, title='')

Write bokeh figure as HTML string to use in `ipyslide.utils.write`.

Parameters

- bokeh_fig : Bokeh figure instance.
- title : Title for figure.

LiveSlides.**cite**(key, citation, here=False)

Add citation in presentation, both key and citation are text/markdown/HTML.

LiveSlides.**clear**()

Clear all slides.

LiveSlides.**clear_notifications**()

Remove all redundent notifications that show up.

LiveSlides.**close_view**()

Close all slides views, but keep slides in memory than can be shown again.

LiveSlides.**colored**(text, fg='blue', bg=None)

Colored text, `fg` and `bg` should be valid CSS colors

LiveSlides.**cols**(*objs, width_percents=None, className=None)

Returns HTML containing multiple columns of given width_percents.

LiveSlides.**convert2slides**(b=False)

Turn ON/OFF slides vs editing mode. Should be in same cell as `LiveSLides`

LiveSlides.**details**(str_html, summary='Click to show content')

Show/Hide Content in collapsed html.

LiveSlides.**doc**(callable, prepend_str=None)

Returns documentation of a callable. You can prepend a class/module name.

LiveSlides.**enable_zoom**(obj)

Add zoom-container class to given object, whether a widget or html/IPYthon object

LiveSlides.**format_css**(selector, **css_props)

Provide CSS values with - replaced by _ e.g. font-size to font_size. selector is a string of valid tag/class/id etc.

LiveSlides.**format_html**(*columns, width_percents=None, className=None)

Same as `write` except it does not write xplicitly, provide in write function

LiveSlides.**html**(tag, children=[], className=None, **node_attrs)

Returns html node with given children and node attributes like style, id etc.

`tag` can be any valid html tag name.

`children` expects:

- str: A string to be added as node's text content.
- html: A html to be added as child node.
- list/tuple of [str, html]: A list of str and html to be added as child nodes.

Example:

```
1  html('img',src='ir_uv.jpg')#Returns IPython.display.HTML("<img src='ir_uv.jpg'></img>") and displas
   image if last line in notebook's cell.
```

LiveSlides.**image**(data=None, width='80%', caption=None, zoomable=True, **kwargs)

Displays PNG/JPEG files or image data etc, `kwrags` are passed to IPython.display.Image.

You can provide following to `data` parameter:

- An opened PIL image. Useful for image operations and then direct writing to slides.
- A file path to image file.
- A url to image file.

- A str/bytes object containing image data.

LiveSlides.**keep_format**(plaintext_or_html)

Bypasses from being parsed by markdown parser. Useful for some graphs, e.g. keep_raw(obj.to_html()) preserves its actual form.

LiveSlides.**notify**(content, title='IPySlides Notification', timeout=5)

Send inside notifications for user to know whats happened on some button click. Set `title = None` if need only content. Remain invisible in screenshot.

LiveSlides.**notify_later**(title='IPySlides Notification', timeout=5)

Decorator to push notification at slide under which it is run.
It should return a string that will be content of notifictaion.
The content is dynamically generated by underlying function,
so you can set timer as well. Remains invisible in screenshot through app itself.

```
@notify_at(title='Notification Title', timeout=5)
def push_notification():
  time = datetime.now()
  return f'Notification at {time}'
```

LiveSlides.**plt2html**(plt_fig=None, transparent=True, caption=None)

Write matplotib figure as HTML string to use in `ipyslide.utils.write`.

Parameters

- plt_fig : Matplotlib's figure instance, auto picks as well.
- transparent: True of False for fig background.
- caption : Caption for figure.

LiveSlides.**pre_compute_display**(b=True)

Load all slides's display and later switch, else loads only current slide. Reset with b = False

This is very useful when you have a lot of Maths or Widgets, no susequent calls to MathJax/Widget Manager required on slide's switch when it is loaded once.

LiveSlides.**print_context**()

Use `print` or function printing with onside in this context manager to display in order.

LiveSlides.**raw**(text)

Keep shape of text as it is, preserving whitespaces as well.

LiveSlides.**refresh**()

Auto Refresh whenever you create new slide or you can force refresh it

LiveSlides.**rows**(*objs, className=None)

Returns tuple of objects. Use in `write`, `iwrite` for better readiability of writing rows in a column.

LiveSlides.**set_dir**(path)

Context manager to set working directory to given path and return to previous working directory when done.

LiveSlides.**show**(fix_buttons=False)

Display Slides. If icons do not show, try with `fix_buttons=True`.

LiveSlides.**sig**(callable, prepend_str=None)

Returns signature of a callable. You can prepend a class/module name.

LiveSlides.**svg**(data=None, caption=None, zoomable=True, **kwargs)

Display svg file or svg string/bytes with additional customizations. `kwrags` are passed to IPython.display.SVG. You can provide url/string/bytes/filepath for svg.

LiveSlides.**textbox**(text, **css_props)

Formats text in a box for writing e.g. inline refrences. `css_props` are applied to box and `-` should be `_` like `font-size` -> `font_size`.

`text` is not parsed to general markdown i.e. only bold italic etc. applied, so if need markdown, parse it to html before. You can have common CSS for all textboxes using class `TextBox`.

LiveSlides.**vspace**(em=1)

Returns html node with given height in em

LiveSlides.**write_citations**(title='### References')

Write all citations collected via `cite` method in the end of the presentation.

LiveSlides.**write_slide_css**(**css_props)

Provide CSS values with - replaced by _ e.g. font-size to font_size.

Content Styling

You can style your content with `className` attribute in writing/content functions. Provide CSS for that using `.format_css` or use some of the available styles. See these styles with `.css_styles` property as below:

Use any or combinations of these styles in className argument of writing functions:

className = 'Center'	------Text------
className = 'Left'	Text----------
className = 'Right'	----------Text
className = 'RTL'	------ اردو عربی
className = 'Info'	Blue Text
className = 'Warning'	Orange Text
className = 'Success'	Green Text
className = 'Error'	Red Text

Loading from File/Other Contexts

LiveSlides.**from_markdown**(path, footer_text='Author Name')

You can create slides from a markdown file or StringIO object as well. It creates slides 1,2,3... in order.

You should add more slides by higher number than the number of slides in the file, or it will overwrite.

Slides separator should be --- (three dashes) in start of line.

Markdown File Content

```
# Talk Title
---
# Slide 1
---
# Slide 2
```

This will create two slides along with title page.

Content of each slide from imported file is stored as list in `slides.md_content`. You can append content to it like this:

```
with slides.slide(2):
  write(slides.md_content[2])
  plot_something()
  write_something()
```

> Note: With this method you can add more slides besides created ones.

LiveSlides.**demo**()

Demo slides with a variety of content.

LiveSlides.**load_docs**()

Create presentation from docs of IPySlides.