IPySlides 4.0.4 Documentation

Creating slides with IPySlides

Abdul Saboor[1], Unknown Author[2]
Jun 04, 2024

[1]My University is somewhere in the middle of nowhere
[2]Their University is somewhere in the middle of nowhere

Table of contents

This is summary of current section

Oh we can use inline columns

Column A

Column B

here and what not!

Markdown

```
```toc Table of contents
Extra content for current section which is on right
```
```

Main App

Slides(extensions=[], auto_focus=True, **settings)

Interactive Slides in IPython Notebook. Only one instance can exist. `auto_focus` can be reset from settings and enable jumping back to slides after a cell is executed. `settings` are passed to `Slides.settings.apply` if you like to set during initialization.

To suppress unwanted print from other libraries/functions, use:

```
with slides.suppress_stdout():
    some_function_that_prints() # This will not be printed
    print('This will not be printed either')
    display('Something') # This will be printed
```

Info

The methods under settings starting with `Slides.settings.set_` returns settings back to enable chaining without extra typing, like `Slides.settings.set_animation().set_layout()...` .

Tip

- Use `Slides.instance()` class method to keep older settings. `Slides()` apply default settings every time.
- Run `slides.demo()` to see a demo of some features.
- Run `slides.docs()` to see documentation.
- Instructions in left settings panel are always on your fingertips

Adding Slides

> **Note**
> Besides functions below, you can add slides with `%%title`/`%%slide` magics as well.

Slides.**title**()

Use this context manager to write title. It is equivalent to `%%title` magic.

Slides.**slide**(slide_number)

Use this context manager to generate any number of slides from a cell. It is equivalent to `%%slide` magic.

> **Info**
> Use this function with '-1' to auto number current slide.

Slides.**frames**(slide_number, iterable, repeat=False)

Decorator for inserting frames on slide, define a function with two arguments (frame_index, frame_content). You can also call it as a function, e.g. `.frames(1,objs)(<optional function>)`.

```
@slides.frames(1,[a,b,c]) # slides 1.1, 1.2, 1.3 with content a,b,c
def f(frame_index, frame_content):
    do_something(frame_content)
    if frame_index == 0: # Main Slide
        print('This is main slide')
```

Python

```python
self.write(self.fmt('`{self.version!r}` `{self.xmd_syntax}`'))
```

'4.0.4'

Extended Markdown

Extended syntax for markdown is constructed to support almost full presentation from Markdown.

Following syntax works only under currently building slide:

- notes`This is slide notes` to add notes to current slide
- cite`key` to add citation to current slide. citations are automatically added in suitable place and should be set once using `Slides.set_citations` function.
- With citations mode set as 'footnote', you can add refs`ncol` to add citations anywhere on slide. If ncol is not given, it will be picked from layout settings.
- section`content` to add a section that will appear in the table of contents.
- toc`Table of content header text` to add a table of contents. For block type toc, see below.
- proxy`placeholder text` to add a proxy that can be updated later with `Slides.get(slide_number).proxies[index].capture` contextmanager or a shortcut `Slides.capture_proxy(slides_number, proxy_index)`. Useful to keep placeholders for plots/widgets in markdwon.
- Triple dashes `---` is used to split markdown text in slides inside `from_markdown(start, content)` function.
- Double dashes `--` is used to split markdown text in frames.

Block table of contents with extra content can be added as follows:

Adding Content

> **Note**
> Besides functions below, you can add content to slides with %%xmd,%xmd as well.

Slides.**write**(*objs, widths=None)

Write `objs` to slides in columns. To create rows in a column, wrap objects in a list or tuple.
You can optionally specify `widths` as a list of percentages for each column.

Write any object that can be displayed in a cell with some additional features:

- Strings will be parsed as as extended markdown that can have citations/python code blocks/Javascript etc.
- Display another function in order by passing it to a lambda function like `lambda: func()`. Only body of the function will be displayed/printed. Return value will be ignored.
- Dispaly IPython widgets such as `ipywidgets` or `ipyvolume` by passing them directly.
- Display Axes/Figure form libraries such as `matplotlib`, `plotly` `altair`, `bokeh`, `ipyvolume` ect. by passing them directly.
- Display source code of functions/classes/modules or other languages by passing them directly or using `Slides.code` API.
- Use `Slides.alt(widget, func)` function to display widget on slides and alternative content in exported slides, function should return possible HTML representation of widget.
- `ipywidgets.HTML` and its subclasses will be displayed as `Slides.alt(widget, html_converter_func)`. The value of exported HTML will be most recent.
- Other options include but not limited to:
  - Output of functions in `ipyslides.utils` module that are also linked to `Slides` object.
  - PIL images, SVGs etc.

Adding Speaker Notes

Skip to Dynamic Content

> **Note**
> You can use notes`notes content` in markdown.

> **Danger**
> This is experimental feature, and may not work as expected.

Slides.notes.**display**()

Slides.notes.**insert**(content)

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

> **Tip**
>
> In markdown, you can use notes`notes content`.

Displaying Source Code

Slides.code.**cast**(obj, language='python', name=None, **kwargs)

Create source code object from file, text or callable. `kwargs` are passed to `ipyslides.formatter.highlight`.

Slides.code.**context**(returns=False, **kwargs)

Execute and displays source code in the context manager. `kwargs` are passed to `ipyslides.formatter.highlight` function. Useful when source is written inside context manager itself. If `returns` is False (by default), then source is displayed before the output of code. Otherwise you can assign the source to a variable and display it later anywhere.

Usage:

```
with source.context(returns = True) as s:
    do_something()
    write(s) # or s.display(), write(s)

#s.raw, s.value are accesible attributes.
#s.focus_lines, s.show_lines are methods that are used to show selective lines.
```

Slides.code.**from_callable**(callable, **kwargs)

Returns source object from a given callable [class,function,module,method etc.] with `show_lines` and `focus_lines` methods. `kwargs` are passed to `ipyslides.formatter.highlight`

Contents

Layout and Theme Settings

Slides.settings.**apply**(**settings)

Apply multiple settings at once. Top level keys should be function names without 'set_' and values should be dictionary of parameters to that function. For example:

```
Slides.settings.apply(
    layout = {"aspect":1.6, "scroll":False},
    footer = {0:"footer text", "numbering":True} # 0 key goes to first positional arg
)
```

Slides.settings.**set_animation**(main='slide_h', frame='appear')

Set animation for slides and frames.

Slides.settings.**set_bg_image**(src=None, opacity=0.25, blur_radius=None)

Adds glassmorphic effect to the background with image. `src` can be a url or a local image path.

Slides.settings.**set_code_theme**(style='default', color=None, background=None, hover_color='var(--hover-bg)', lineno=True)

Set code style CSS. Use background for better view of your choice. This is overwritten by theme change.

Slides.settings.**set_css**(props: dict)

Set CSS for all slides. This loads on slides navigation, so you can include keyframes animations as well.

Useful Functions for Rich Content

Slides.**clip_image**(filename, quality=95, overwrite=False)

Save image from clipboard to file with a given quality. On next run, it loads from saved file under `notebook-dir/.ipyslides-assets/clips`. Useful to add screenshots from system into IPython. You can use overwite to overwrite existing file. You can add saved clips using a "clip:" prefix in path in `Slides.image("clip:filename.png")` function and also in markdown.

- Output can be directly used in `write` command.
- Converts to PIL image using `.to_pil()`.
- Convert to HTML representation using `.to_html()`.
- Convert to Numpy array using `.to_numpy()` in RGB format that you can plot later.

Slides.**alt**(widget, func)

Display `widget` for slides and output of `func(widget)` will be and displayed only in exported formats as HTML. `func` should return possible HTML representation (provided by user) of widget as string.

Python

```python
import ipywidgets as ipw
slides = get_slides_instance()
slides.alt(ipw.IntSlider(),lambda w: f'<input type="range" min="{w.min}" max="{w.max
```

Info

Citations and Sections

Use syntax cite`key` to add citations which should be already set by `Slides.set_citations(data, mode)` method. Citations are written on suitable place according to given mode. Number of columns in citations are determined by `Slides.settings.set_layout(..., ncol_refs = int)`. [1]

Add sections in slides to separate content by section`text`. Corresponding table of contents can be added with toc`title`/```toc title\n summary of current section \n```.

Slides.**set_citations**(data, mode='footnote')

Set citations from dictionary or file that should be a JSON file with citations keys and values, key should be cited in markdown as cite`key`. `mode` for citations should be one of ['inline', 'footnote']. Number of columns in citations are determined by `Slides.settings.set_layout(..., ncol_refs=N)`.

> **Note**
>
> - You should set citations in start if using voila or python script. Setting in start in notebook is useful as well.
> - Citations are replaced with new ones, so latest use of this function reprsents avilable citations.

1. Citation A

Dynamic Content

Slides.**on_refresh**(func)

Decorator for inserting dynamic content on slide, define a function with no arguments. Content updates when `slide.update_display` is called or when `Slides.refresh` is called.

> 💡 Tip
>
> You can use it to dynamically fetch a value from a database or API while presenting, without having to run the cell again.

> 📝 Note
>
> - No return value is required. If any, should be like `display('some value')`, otherwise it will be ignored.
> - A slide with dynamic content enables a refresh button in bottom bar.
> - All slides with dynamic content are updated when refresh button in top bar is clicked.

Python

```python
import time
slides = get_slides_instance() # Get slides instance, this is to make doctring runn
source.display() # Display source code of the block
@slides.on_refresh
def update_time():
    print('Local Time: {3}:{4}:{5}'.format(*time.localtime())) # Print time in HH:M
# Updates on update_display or refresh button click
```

Content Styling

You can **style** or **colorize** your *content* and *text*. 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	Formatting Style
'text-[value]'	[value] should be one of tiny, small, big, large, huge.
'align-[value]'	[value] should be one of center, left, right.
'rtl'	———— اردو عربی
'info'	Blue text. Icon ℹ️ for note-info class.
'tip'	Blue Text. Icon 💡 for note-tip class.
'warning'	Orange Text. Icon ⚠️ for note-warning class.
'success'	Green text. Icon ✅ for note-success class.
'error'	Red Text. Icon ⚡ for note-error class.
'note'	📝 Text with note icon.
'export-only'	Hidden on main slides, but will appear in exported slides.
'jupyter-only'	Hidden on exported slides, but will appear on main slides.
'block'	Block of text/objects
'block-[color]'	Block of text/objects with specific background color from red, green, blue, yellow, cyan, magenta and gray.

Python

```python
self.write(('You can **style**{.error} or **color[teal]`colorize`** your *content*{
        'Provide **CSS**{.info} for that using `.format_css` or use some of the avai
        'See these **styles**{.success} with `.css_styles` property as below:'))
```

Highlighting Code

pygments is used for syntax highlighting [1]. You can **highlight** code using `highlight` function [2] or within markdown like this:

Python

```python
import ipyslides as isd
```

Javascript

```javascript
import React, { Component } from "react";
```

Markdown

```markdown
## Highlighting Code
[pygments](https://pygments.org/) is used for syntax highlighting cite`A`.
You can **highlight**{.error} code using `highlight` function cite`B` or within m
```python
import ipyslides as isd
```
```javascript
import React, { Component } from "react";
```
proxy`source code of slide will be updated here later using slide_handle.proxies
```

1. Citation A

2. Citation B

Loading from File/Exporting to HTML

> **Note**
> You can parse and view a markdown file. The output you can save by exporting notebook in other formats.

Slides.**sync_with_file**(start, path, trusted=False, interval=500)

Auto update slides when content of markdown file changes. You can stop syncing using `Slides.unsync` function. interval is in milliseconds, 500 ms default. Read `Slides.from_markdown` docs about content of file.

The variables inserted in file content are used from top scope.

Slides.**from_markdown**(start, content, trusted=False)

You can create slides from a markdown tex block as well. It creates slides `start + (0,1,2,3...)` in order. You should add more slides by higher number than the number of slides in the file/text, or it will overwrite.

- Slides separator should be --- (three dashes) in start of line.
- Frames separator should be -- (two dashes) in start of line. All markdown before first `--` will be written on all frames.
- In case of frames, you can add %++ (percent plus plus) in the content to add frames incrementally.
- You can use frames separator (--) inside `multicol` to make columns span multiple frames with %++.

Markdown content of each slide is stored as `.markdown` attribute to slide. You can append content to it later like this:

Contents

Adding User defined Objects/Markdown Extensions

I will be on exported slides

Python

```python
self.write('## Adding User defined Objects/Markdown Extension
self.write(
    lambda: display(self.html('h3','I will be on main slides'
    metadata = {'text/html': '<h3 class="warning">I will be o
    s.get_source(), widths = [1,3]
)
self.write('If you need to serialize your own or third party
self.doc(self.serializer,'Slides.serializer', members = True,
self.write('**You can also extend markdown syntax** using `ma
self.doc(self.extender,'Slides.extender', members = True, its
```

> **Note**
> If you need to serialize your own or third party objects not serialized by this module, you can use `@Slides.serializer.register` to serialize them to html.

Slides.serializer.**display**(obj)

Display an object with metadata if a serializer available. Same as display(obj, metadata = serializer.get_metadata(obj)))

Slides.serializer.**get_func**(obj_type)

Get serializer function for a type. Returns None if not found.

Focus on what matters

- There is a zoom button on top bar which enables zooming of certain elements. This can be toggled by `Z` key.
- Most of supported elements are zoomable by default like images, matplotlib, bokeh, PIL image, altair plotly, dataframe, etc.
- You can also enable zooming for an object/widget by wrapping it inside `Slide.`

  function conveniently. - You can also enable by manully adding

  `zoom-self`, `zoom-child` classes to an element. To prevent zooming under as `zoom-child` class, use `no-zoom` class.

Focus on Me 😎

- If zoom button is enabled, you can hover here to zoom in this part!
- You can also zoom in this part by pressing `Z` key while mouse is over this part.

SVG Icons

Icons that apprear on buttons inslides (and their rotations) available to use in your slides as well

chevron: pencil: bars: arrow: arrow-bar: close: dots: expand: compress: camera: play: pause: stop: loading: circle: info: refresh: laser: zoom-in: zoom-out: search: code: win-maximize: win-restore: rows: columns: settings:

Python

```python
import ipywidgets as ipw
btn = ipw.Button(description='Chevron-Down',icon='plus').add_class('MyIcon') # Any j
self.write(btn)
self.format_css({'.MyIcon .fa.fa-plus': self.icon('chevron',color='crimson', size='1
```

Auto Slide Numbering

Use -1 as placeholder to update slide number automatically.

- In Jupyter notebook, this will be updated to current slide number.
- In python file, it stays same.
- You need to run cell twice if creating slides inside a for loop while using `-1`.

Presentation Code

Python

```python
def docs(self):
    "Create presentation from docs of IPySlides."
    self.close_view() # Close any previous view to speed up loading 10x faster on a
    self.clear() # Clear previous content
    self.create(*range(22)) # Create slides faster

    from ..core import Slides

    self.set_citations({'A': 'Citation A', 'B': 'Citation B'}, mode = 'footnote')
    self.settings.set_footer('IPySlides Documentation')

    with self.title(): # Title
        self.write(f'## IPySlides {self.version} Documentation\n### Creating slides
        self.center('''
            alert`Abdul Saboor`sup`1`, Unknown Authorsup`2`
```