IPySlides 4.3.1 Documentation

Creating slides with IPySlides

Abdul Saboor[1]

Jun 18, 2024

[1]My 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.display` whenever possible instead of IPython's display, it automatically adds serializer metadat.
> - 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

Adding Slides

> 📝 Note
> Besides function below, you can add slides with `%%slide number [-m]` magic as well.

Slides.**build**(slide_number, /, content=None, *, repeat=False, trusted=False)

Build slides with a single unified command in three ways:

- `slides.build(number, str)` creates many slides with markdown content.
  - Equivalent to `%%slide number -m` magic in case of one slide.
  - Frames separator is double dashes `--` and slides separator is triple dashes `---`. Same applies to `Slides.sync_with_file` too.
  - Markdown before the first `--` (frame separator) is written on all frames.
  - Use %++ in frames to add frames incrementally.
  - See `slides.xmd_syntax` for extended markdown usage.
  - Keyword argument `trusted` is used here if there are `python run` blocks in markdown.
- `with slides.build(number):` creates single slide. Equivalent to `%%slide number` magic.
- `@slides.build(number, iterable)` creates a slide with multiple frames.
  - `iterable` should be list-like object. Any level of nesting should be handled by `func`.
  - Automatic call as `slides.build(number, iterable)()` will write objects from top to bottom.
  - Use decorated `func(frame_index, frame_content)` to write content flexibly.
  - `repeat` can be False or True to enable enable making `iterable` a grid. Top list-like creates frames, inner list-like items can be used to create columns and automatically written if called as `Slides.build(...)()`. In case of `repeate = True`, you can loop over rows yourself as well to write columns with given widths in `write` command.
  - If function has a docstring, it will be parsed and added on top of all frames.
  - Use voffetinteger in px in markdown/docstring or `Slides.this.voffset(integer)` to

Python

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

'4.3.1'

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[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 text in slides inside markdown content of `Slides.build` function or markdown file.
- Double dashes `--` is used to split 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` function to display obj/widget on slides and alternative content in exported slides.
- Use `Slides.alt_clip` function to display anything (without parsing) on slides and paste its screenshot for export. Screenshots are persistent and taken on slides.
- Use `Slides.image_clip` to add screenshots from clipboard while running the cell.
- `ipywidgets.[HTML, Output, Box]` and their subclasses will be displayed as `Slides.alt(html_converter_func, widget)`. The value of exported HTML will be most recent

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, filter='blur(2px)', contain=False)

Adds glassmorphic effect to the background with image. `src` can be a url or a local image path. Overall background will not be exported, but on each slides will be. This is to keep exported file size minimal.

Slides.settings.**set_code_theme**(style='default', color=None, background=None, hover_color='var(--alternate-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)

Useful Functions for Rich Content

Slides.**alt**(func_or_html, obj, /)

Display `obj` for slides and output of `func_or_html` will be and displayed only in exported formats as HTML.

- `func_or_html` should be a `str`, an obj with `_repr_html_` method or a callable to receive `obj` as its only argument.
- In case `obj` is an instance of `ipywidgets.DOMWidget`:
  - A callable `func_or_html` will give the latest representation of widget in exported slides.
  - In other cases, it will export the runtime representation of widget.
- For any other `obj`, representation is always computed at runtime.

Python

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

Info

- If you happen to be using `alt` many times for same type, you can use `Slides.serializer.register` and then pass that type of widget without alt.
- `ipywidgets`'s `HTML`, `Box` and `Output` widgets and their subclasses directly give html representation if used inside `write` command.

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 runna
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:MI
# 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 css_class argument of writing functions:
```

css_class	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 ℹ 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_slide_number, /, 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.build` docs about content of file.

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

Slides.**demo**()

Demo slides with a variety of content.

Slides.**docs**()

Create presentation from docs of IPySlides.

Slides.**export_html**(path='slides.html', overwrite=False)

Build beautiful html slides that you can print.

- Use 'overrides.css' file in same folder to override CSS styles.
- If a slide has only widgets or does not have single object with HTML representation, it will be skipped.
- You can take screenshot (using system's tool) of a widget and add it back to slide using

Contents

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o != ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o != ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0	1

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o != ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0

1

2

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o ≠ ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0	1
2	3

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o ≠ ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0	1
2	3
4	

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o ≠ ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0	1
2	3
4	5

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o ≠ ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0	1
2	3
4	5	6

Adding content on frames incrementally

Python

```python
@self.build(-1, [(0,1), (2,3),(4,5,6,7)], repeat=True)
def make_frames(idx, obj):
    "# Adding content on frames incrementally yoffset`5`"
    code.focus_lines([o for ob in obj for o in ob if o ≠ ''']).display() # flatten
    for ws, cols in zip([None, (2,3),None],obj):
        cols = [self.html('h1', f"{c}",
            style="background:var(--alternate-bg);margin-block:4px !important;") fo
        self.write(*cols, widths=ws)
```

0	1
2	3
4	5	6	7

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 besides standard ipywidgets icons.

arrow: arrowb: arrowbd: arrowbl: arrowbr: arrowbu: arrowd: arrowl: arrowr: arrowu: bars: camera: chevron: chevrond: chevronl: chevronr: chevronu: circle: close: code: columns: compress: dots: edit: expand: info: laser: loading: pause: pencil: play: refresh: rows: search: settings: stop: win-maximize: win-restore: zoom-in: zoom-out:

Python

```python
import ipywidgets as ipw
btn = ipw.Button(description='Chevron-Down Icon',icon='chevrond')
self.write(btn)
```

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`.
- Additionally, in python file, you can use `Slides.build_` instead of 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(23)) # Create slides faster

    from ..core import Slides

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

    with self.build(0): # Title page
        self.this.set_bg_image(Path(__file__).parent.parent.parent / 'slide.png',1,
        self.write(f'## IPySlides {self.version} Documentation\n### Creating slides
        self.center(self.fmt('''
```