

# IPySlides 4.7.0 Documentation

## Creating slides with IPySlides

Abdul Saboor<sup>1</sup>

Jul 25, 2024



<sup>1</sup>My University is somewhere in the middle of nowhere

## Table of contents

1. | **Introduction**
2. Adding Slides and Content
3. Layout and **Theme** Settings
4. Useful Functions for **Rich Content**
5. Loading from File/Exporting to HTML
6. Advanced Functionality
7. Presentation Code

## This is summary of current section

Oh we can use inline columns

Column A

Column B

here and what not!



### Markdown

```
1  ```multicol .block-green
2  toc[True]`## Table of contents`
3  +++
4  Extra content for current section which is on right
5  ```
```

# Main App

**Slides**(extensions=[], \*\*settings)

Interactive Slides in IPython Notebook. Only one instance can exist. settings are passed to `Slides.settings()` if you like to set during initialization.

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

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



## Info

The traitlets callables under settings returns settings back to enable chaining without extra typing, like `Slides.settings.logo().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.
- Creating slides in a batch using `Slides.create` is much faster than adding them one by one.
- In JupyterLab, right click on the slides and select **Create New View for Output** for optimized display.

# Adding Slides



## Note

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

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

Build slides with a single unified command in three ways:

1. `slides.build(number, str, trusted)` 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.
  - Use `%++` to join content of frames incrementally.
  - Markdown multicol before `--` creates incremental columns if `%++` is provided.
  - See `slides.xmd_syntax` for extended markdown usage.
  - Keyword argument `trusted` is used here if there are python run blocks in markdown.
  - To debug markdown content, use EOF on its own line to keep editing and clearing errors. Same applies to `Slides.sync_with_file` too.
2. `slides.build(number, list/tuple, widths)` to create a slide from list-like contents immediately.
  - We use `write(*contents, widths)` to make slide. This is a shortcut way of step 3 if you want to create slides fast with few objects.
3. **with** `slides.build(number)`: creates single slide. Equivalent to `%%slide number` magic.
  - Use `fsep()` from top import or `Slides.fsep()` to split content into frames.
  - Use **for** item **in** `fsep.loop(iterable)`: block to automatically add frame separator.

# Important Methods on Slide

## Alert

Use slide handle or `Slides[number, ]` to apply these methods because index can change on new builds.

**`Slide.yoffset(value)`**

Set yoffset (in percent) for frames to have equal height in incremental content.

**`Slide.set_animation(this=None, main=None, frame=None)`**

Set animation of this slide. Provide None if need to stop animation. Use `main_all` and `frame` to set animation to all slides.

**`Slide.set_bg_image(src=None, opacity=1, filter=None, contain=False)`**

Adds background image to this slide. `src` can be a url or a local image path or an svg str. `filter` is a CSS filter like `blur(5px)`, `grayscale()` etc.

## Tip

This function enables you to add a slide purely with an image, possibly with `opacity=1` and `contain = True`.

**`Slide.update_display(go_there=True)`**

Update display of this slide.

**`Slide.get_source(name=None)`**

```
1 self.write(self.fmt('{self.version!r}' '{self.xmd_syntax}', self=self))
```

'4.7.0'

## 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 using `with` `Slides[slide_number,].proxies[index]:` or a shortcut `with` `Slides.capture_proxy(slides_number, proxy_index):`. Useful to keep placeholders for plots/widgets in markdown.
- 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. Alongwith this `%++` can be used to increment text on framed slide.

Block table of contents with extra content as summary of current section 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, css_class=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. `css_class` can have multiple classes separated by space, works only for multiple columns.

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

- Strings will be parsed 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.
- Display IPython widgets such as `ipywidgets` or `ipyvolume` by passing them directly.
- Display Axes/Figure from 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/screenshot in exported slides.
- Use `Slides.clip` to add screenshots from clipboard.
- `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](#)

You can use `notes`notes content`` in markdown.\n{.note .success}\n

## 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:

```
1 with source.context(returns = True) as s:
2     do_something()
3     write(s) # or s.display(), write(s)
4
5 #s.raw, s.value are accesible attributes.
6 #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

1. Introduction
2. Adding Slides and Content
- 3. Layout and Theme Settings**
4. Useful Functions for Rich Content
5. Loading from File/Exporting to HTML
6. Advanced Functionality
7. Presentation Code

# Layout and Theme Settings

## Slides.Settings

Apply settings to slides programatically. Fewer settings are available as widgets.

Settings can be nested or individual attributes as set as well. For example:

```
1 Slides.settings(layout = {"aspect": 16/10}) # Top
2 Slides.settings.layout(aspect = 16/10) # Individual
3 Slides.settings.layout.aspect = 16/10 # Attribute
```

All settings calls including top level returns settings instance to apply method chaining. e.g.

```
Slides.settings.layout(aspect = 16/10).footer(text="ABC").logo( ... ).
```

## Slides.Settings.Code

Set code block styles. background and color may be needed for some styles.

## Slides.Settings.Fonts

Set fonts of text and code and size.

## Slides.Settings.Footer

Set footer attributes of slides.

## Slides.Settings.Layout

Set layout of slides.

# Useful Functions for Rich Content

Font Size



18

clipboard image

Markdown

```
1 clip[caption=clipboard image]`test.png`
```

`Slides.alt(exportable_data, obj, /, **kwargs)`

Display `obj` for slides and output of `exportable_data` will be and displayed only in exported formats as HTML. `kwargs` are passed to `clip` when `exportable_data` is an image filename such as `'clip:test.png'` to save clipboard image.

- `exportable_data` should be an html str or an image file name prefixed with 'clip' e.g. `'clip:filename.png'` or a callable to receive `obj` as its only argument.
  - A callable will give the latest representation of widget in exported slides and runtime representation of any other `obj`.
  - An html str, it will export the runtime representation of `obj`.
  - A `'clip:filename.png'`, will allow to take screenshot on slides for export, this may be most flexible use case.



Python

```
1 import ipywidgets as ipw
2 slides = get_slides_instance()
3 slides.alt(lambda w: f'<input type="range" min="{w.min}" max="{w.max}" value="{w.value}">', i
```

# 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.layout(..., ncol_refs = int)`.<sup>1</sup>

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

`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.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 represents available citations.

---

<sup>1</sup>. Citation A

# Dynamic Content

```
Slides.interact(_BaseSlides__func=None, _BaseSlides__options={'manual': True, 'height': ''},  
**kwargs)
```

ipywidgets's interact functionality tailored for ipyslides's needs. It adds 'height' as additional parameter in options. Set height to avoid flickering output.

Python

```
1 import time  
2 slides = get_slides_instance() # Get slides instance, this is to make docstring runnable  
3 source.display() # Display source code of the block  
4 @slides.interact({'height':'2em'}, date = False)  
5 def update_time(date):  
6     local_time = time.localtime()  
7     objs = ['Time: {3}:{4}:{5}'.format(*local_time)] # Print time in HH:MM:SS format  
8     if date:  
9         objs.append('Date: {0}/{1}/{2}'.format(*local_time))  
10    slides.write(*objs)
```

Time: 13:4:50









Tip

You can use this inside columns using delayed display trick, like `write('First column', lambda: interact(f,`

# Content Styling

You can **style** or **colorize** your *content* and *text*. Provide **CSS** for that using `Slides.html("style", ...)` or use some of the available styles. See these **styles** with `Slides.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  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.
'raw-text'	Text will not be formatted and will be shown as it is.
'zoom-self'	Zooms object on hover, when Zoom is enabled.
'zoom-child'	Zooms child object on hover, when Zoom is enabled.

Python

# Highlighting Code

[pygments](#) is used for syntax highlighting <sup>1</sup>. You can **highlight** code using highlight function <sup>2</sup> or within markdown like this:

Python

```
1 import ipyslides as isd
```

Javascript

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



Markdown

```
1 ## Highlighting Code
2 [pygments](https://pygments.org/) is used for syntax highlighting cite`A`.
3 You can **highlight**{.error} code using `highlight` function cite`B` or within markdown 1:
4 ```python
5 import ipyslides as isd
6 ```
7 ```javascript
8 import React, { Component } from "react";
9 ```
10 proxy`source code of slide will be updated here later using slide_handle.proxies[0] contex
```



# Loading from File/Exporting to HTML



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.



To debug a linked file, use EOF on its own line to keep editing and clearing errors.

**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 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.

# Contents

1. Introduction
2. Adding Slides and Content
3. Layout and **Theme** Settings
4. Useful Functions for **Rich Content**
5. Loading from File/Exporting to HTML
- 6. Advanced Functionality**
7. Presentation Code

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

0

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

0

1

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

0

1

2

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

0

1

2

3

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

0

1

2

3

4



# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

0

1

2

3

4

5

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    self.write(*cols, widths=ws)
```

0

1

2

3

4

5

6

# Adding content on frames incrementally

Python

```
1 self.write("## Adding content on frames incrementally yoffset`0`")
2 self.frozen(widget := (code := s.get_source()).as_widget()).display()
3 self.fsep() # frozen in above line get oldest metadata for export
4 def highlight_code(slide): widget.value = code.focus_lines(range(slide.indexf + 1)).value
5 self.on_load(highlight_code)
6
7 for ws, cols in self.fsep.loop(zip([None, (2,3),None], [(0,1),(2,3),(4,5,6,7)])):
8     cols = [self.html('h1', f"{c}", style="background:var(--bg3-color);margin-block:0.05em !im
9     self.fsep.join() # incremental
10    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

```
1 self.write('## Adding User defined Objects/Markdown Extensions')
2 self.write(
3     lambda: display(self.html('h3','I will be on main slides',css_cl
4     metadata = {'text/html': '<h3 class="warning">I will be on expor
5     s.get_source(), widths = [1,3]
6 )
7 self.write('If you need to serialize your own or third party objects
8 self.doc(self.serializer,'Slides.serializer', members = True, itself
9 self.write('**You can also extend markdown syntax** using `markdown
10 self.doc(self.extender,'Slides.extender', members = True, itself = F
```



## 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.get\_func(obj\_type)**

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

**Slides.serializer.get\_html(obj\_type)**

Get html str of a registered obj\_type.

## 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.zoomable`` function conveniently.
- You can also enable by manually adding `zoom-self`, `zoom-child` classes to an element. To prevent zooming under a `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 appear 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: > chevrong: ∨ 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

```
1 import ipywidgets as ipw
2 btn = ipw.Button(description='Chevron-Down Icon', icon='chevrong')
3 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

```
1  def docs(self):
2      "Create presentation from docs of IPySlides."
3      self.close_view() # Close any previous view to speed up loading 10x faster on average
4      self.clear() # Clear previous content
5      self.create(range(24)) # Create slides faster
6
7      from ..core import Slides
8
9      self.set_citations({'A': 'Citation A', 'B': 'Citation B'}, mode = 'footnote')
10     self.settings.footer(text='IPySlides Documentation', date=None)
11
12     with self.build(0): # Title page
13         self.this.set_bg_image(self.get_logo(), 0.25, filter='blur(10px)', contain=True)
14         self.write(f'## IPySlides {self.version} Documentation\n### Creating slides with IPyS
15         self.center(self.fmt(''
16             alert`Abdul Saboor`sup`1`
```