

Welcome To Presentate!

Tools for creating slides.

By @pacaunt

Contents

1	Introduction	3
1.1	What is Presentate?	3
1.2	Motivation	4
1.3	Usage	5
2	Features	6
2.1	Simple Animations	6
2.2	Mathematical Equation Animation	7
2.3	Cover Functions	8
2.4	CeTZ Support	9
2.5	Works with Pinit	10
2.6	Fletcher Support	12

2.7	Fake Frozen Counters	13
2.8	Handout and Drafted Modes	14
2.9	Pdfpc Support	15
3	Internals	16
4	List of Available Functions	19
4.1	Presentate Module	19
4.2	Animation Module	20
4.3	Themes Module	23
4.4	Store Module	24

1 Introduction

1.1 What is Presentate?

It is a simple, minimal tool created in pure Typst for creating slides.

It packed with simple animation like `pause` and `meanwhile`, to complex `only` and `uncover`.

You may try `#pause` to find out that this section comes later.

1.1 What is Presentate?

It is a simple, minimal tool created in pure Typst for creating slides.

It packed with simple animation like `pause` and `meanwhile`, to complex `only` and `uncover`.

You may try `#pause` to find out that this section comes later.

I came later!

1.2 Motivation

I am an average undergraduate student that has an impression to create a presentation in Typst. However, the existing package does not suit my needs, as I wish more flexibility to customize the cover functions (animations) and page configurations. So I write some code with some hacks to create this package.

Big thanks to Touying and Polylux creators that inspired me this package, some parts of this code even came from them.

One big flaw of this package is that, it requires very long compilation time. So, choose the one that suits your needs!

1.3 Usage

Just import the module,

```
#import "@preview/presentate:0.1.0": *  
  
#slide[  
  = Hello World!  
]
```

and begin your journey.

2 Features

2.1 Simple Animations

`#pause` is used to show the content incrementally. Like this:

`#meanwhile` Meanwhile, `#pause #meanwhile` is used to show the content `#pause`
`_parallel_` to `#pause`.

Meanwhile,

2.1 Simple Animations

`#pause` is used to show the content incrementally. Like this:

- First `#pause`
- Second `#pause`
- Third `#pause`

- First

`#meanwhile` Meanwhile, `#pause` `#meanwhile` is used to show the content `#pause` `_parallel_` to `#pause`.

Meanwhile, `#meanwhile` is used to show the content

2.1 Simple Animations

`#pause` is used to show the content incrementally. Like this:

```
- First #pause  
- Second #pause  
- Third #pause
```

- First
- Second

`#meanwhile` Meanwhile, `#pause` `#meanwhile` is used to show the content `#pause` `_parallel_` to `#pause`.

Meanwhile, `#meanwhile` is used to show the content *parallel* to `#pause`.

2.1 Simple Animations

`#pause` is used to show the content incrementally. Like this:

```
- First #pause  
  
- Second #pause  
  
- Third #pause
```

- First
- Second
- Third

`#meanwhile` Meanwhile, `#pause #meanwhile` is used to show the content `#pause` `_parallel_` to `#pause`.

Meanwhile, `#meanwhile` is used to show the content *parallel* to `#pause`.

2.1 Simple Animations

`#pause` is used to show the content incrementally. Like this:

```
- First #pause  
  
- Second #pause  
  
- Third #pause
```

- First
- Second
- Third

`#meanwhile` Meanwhile, `#pause` `#meanwhile` is used to show the content `#pause` `_parallel_` to `#pause`.

Meanwhile, `#meanwhile` is used to show the content *parallel* to `#pause`.

2.2 Mathematical Equation Animation

`#pause` and `#meanwhile` also can be used with `#math.equation`:

```
$  
f(x) &= (x + 1)^2 pause \  
      &= (x + 1)(x + 1) pause \  
      &= x^2 + 2x + 1  
$
```

Results:

$$f(x) = (x + 1)^2$$

2.2 Mathematical Equation Animation

`#pause` and `#meanwhile` also can be used with `#math.equation`:

```
$  
f(x) &= (x + 1)^2 pause \  
      &= (x + 1)(x + 1) pause \  
      &= x^2 + 2x + 1  
$
```

Results:

$$\begin{aligned} f(x) &= (x + 1)^2 \\ &= (x + 1)(x + 1) \end{aligned}$$

2.2 Mathematical Equation Animation

`#pause` and `#meanwhile` also can be used with `#math.equation`:

```
$  
f(x) &= (x + 1)^2 pause \  
      &= (x + 1)(x + 1) pause \  
      &= x^2 + 2x + 1  
$
```

Results:

$$\begin{aligned} f(x) &= (x + 1)^2 \\ &= (x + 1)(x + 1) \\ &= x^2 + 2x + 1 \end{aligned}$$

2.3 Cover Functions

We have `#uncover` and `only` for show the content in some specific subslides. `#only` does not reserve space, but `#uncover` reserves space.

```
Hello #only(1, 3, text(fill: red)[
  Only at subslide 2!
]) There!
```

```
Uncover #uncover(from: 2,  
    text(fill: green)[  
        from subslide 2  
    ]) and then on. #pause See?
```

Results:

Hello **Only** at subslide 2! There!

Uncover and
then on.

2.3 Cover Functions

We have `#uncover` and `only` for show the content in some specific subslides. `#only` does not reserve space, but `#uncover` reserves space.

```
Hello #only(1, 3, text(fill: red)[
  Only at subslide 2!
]) There!
```

```
Uncover #uncover(from: 2,
  text(fill: green)[
    from subslide 2
  ]) and then on. #pause See?
```

Results:

Hello There!

Uncover from subslide 2 and then on. See?

2.3 Cover Functions

We have `#uncover` and `only` for show the content in some specific subslides. `#only` does not reserve space, but `#uncover` reserves space.

```
Hello #only(1, 3, text(fill: red)[
  Only at subslide 2!
]) There!
```

```
Uncover #uncover(from: 2,
  text(fill: green)[
    from subslide 2
  ]) and then on. #pause See?
```

Results:

Hello **Only at subslide 2!** There!

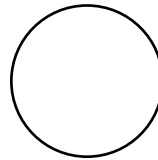
Uncover **from subslide 2** and then on. See?

2.4 CeTZ Support

```
#import "@preview/cetz:0.3.4": canvas, draw
#let cz = animate(
  cover: draw.hide.with(bounds: true),
  marker: draw.content.with(()),
  clear: draw.hide,
)

#context canvas({
  import draw: *
  circle((0, 0))
  content(), pause
  circle((1, 0))
  (cz.uncover)(3, rect((2, -1), (4, 1)))
})
```

`#pause` and `#meanwhile` are natively usable with CeTZ. You can use `#animate` constructor to create functions that suit for CeTZ environment. Note the `#context {}`. Results:

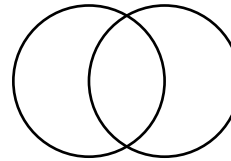


2.4 CeTZ Support

```
#import "@preview/cetz:0.3.4": canvas, draw
#let cz = animate(
  cover: draw.hide.with(bounds: true),
  marker: draw.content.with(()),
  clear: draw.hide,
)

#context canvas({
  import draw: *
  circle((0, 0))
  content(), pause
  circle((1, 0))
  (cz.uncover)(3, rect((2, -1), (4, 1)))
})
```

`#pause` and `#meanwhile` are natively usable with CeTZ. You can use `#animate` constructor to create functions that suit for CeTZ environment. Note the `#context {}`. Results:

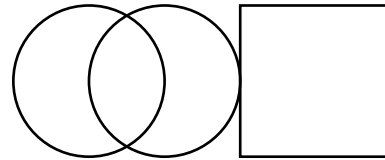


2.4 CeTZ Support

```
#import "@preview/cetz:0.3.4": canvas, draw
#let cz = animate(
  cover: draw.hide.with(bounds: true),
  marker: draw.content.with(()),
  clear: draw.hide,
)

#context canvas({
  import draw: *
  circle((0, 0))
  content(), pause
  circle((1, 0))
  (cz.uncover)(3, rect((2, -1), (4, 1)))
})
```

`#pause` and `#meanwhile` are natively usable with CeTZ. You can use `#animate` constructor to create functions that suit for CeTZ environment. Note the `#context {}`. Results:



2.5 Works with Pinit

Pythagorean theorem:

$$a^2 + b^2 = c^2$$

2.5 Works with Pinit

Pythagorean theorem:

$$a^2 + b^2 = c^2$$

a^2 and b^2 : squares of triangle legs

2.5 Works with Pinit

Pythagorean theorem:

$$a^2 + b^2 = c^2$$

a^2 and b^2 : squares of triangle legs

c^2 : square of hypotenuse

larger than a^2 and b^2



The source code is

```
#import "@preview/pinit:0.2.2": *
Pythagorean theorem:
$ #pin(1)a^2#pin(2) + #pin(3)b^2#pin(4) = #pin(5)c^2#pin(6) $
  #pause
$a^2$ and $b^2$ : squares of triangle legs
#only(2, {
  pinit-highlight(1,2)
  pinit-highlight(3,4)
})
  #pause

$c^2$ : square of hypotenuse
#pinit-highlight(5,6, fill: green.transparentize(80%))
#pinit-point-from(6)[larger than $a^2$ and $b^2$]
```

2.6 Fletcher Support

```
#import "@preview/fletcher:0.5.8" as
fletcher: diagram, node, edge

#let new-diagram = reducer.with(cover:
fletcher.hide, func: diagram)
#let ft = animate(cover: fletcher.hide,
combine: (it, mark) => (it, mark))

#context new-diagram(
  node((0, 0), [Start]),
  pause,
  edge("d", "->"),
  node((0, 1), [End]),
  ..(ft.uncover)(2, edge("d", "->")),
  pause,
  node((0, 2), [Longer End.]),
)
```

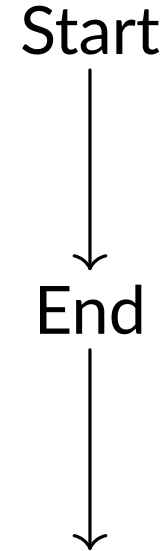
Start

2.6 Fletcher Support

```
#import "@preview/fletcher:0.5.8" as
fletcher: diagram, node, edge

#let new-diagram = reducer.with(cover:
fletcher.hide, func: diagram)
#let ft = animate(cover: fletcher.hide,
combine: (it, mark) => (it, mark))

#context new-diagram(
  node((0, 0), [Start]),
  pause,
  edge("d", "->"),
  node((0, 1), [End]),
  ..(ft.uncover)(2, edge("d", "->")),
  pause,
  node((0, 2), [Longer End.]),
)
```

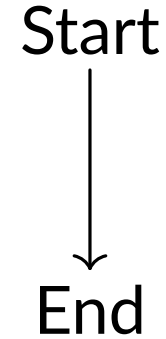


2.6 Fletcher Support

```
#import "@preview/fletcher:0.5.8" as
fletcher: diagram, node, edge

#let new-diagram = reducer.with(cover:
fletcher.hide, func: diagram)
#let ft = animate(cover: fletcher.hide,
combine: (it, mark) => (it, mark))

#context new-diagram(
  node((0, 0), [Start]),
  pause,
  edge("d", "->"),
  node((0, 1), [End]),
  ..(ft.uncover)(2, edge("d", "->")),
  pause,
  node((0, 2), [Longer End.]),
)
```



Longer End.

2.7 Fake Frozen Counters

As you can see from our slides. It has correct page number and heading number.

By default `#heading`, `#figure`, `#quote`, `#table` and `#math.equation` counters are frozen.

However, we done this by calculation and placing `#alias-counter` to rewind the counter. Therefore, if you have manual updates, Presentate will **not** see it.

2.8 Handout and Drafted Modes

You can set options of your presentation by `#set-options(. .)`.

The available options are:

1. `handout`: (bool) disabling all animations.
2. `drafted`: (bool) placing subslide number on the slides.
3. `freeze-counter` (bool) freezing the counters.

2.9 Pdfpc Support

You can use Polylux's `polylux2pdfpc` in command line to generate a `.pdfpc` file of this presentation.

Then, Pdfpc will recognize the `.pdfpc` file and the overlays will be supported.

Currently, only overlays in pdfpc are supported.

3 Internals

Presentate uses some counters and states along with many complex show rules. Thus, each `#pause` and `#meanwhile` are costly, as they require multiple compilations.

State variables in Presentate are stored in `store.typ`, you can access by `#store.subslides.get()` to retrieve current subslide, and `#store.dynamics.get()` to retrieve the number of `#pause` and update maximal number of subslides created as `store.dynamics` stores a dictionary (`pause: int, steps: int`) where `pause` key is current number of pause, and `steps` is maximum subslides needed.

Frozen Counters are able to do because Presentate creates an `#alias-counter` to count the presence of respective elements in the first subslide, and update the *real* counter by subtracting it.

Therefore, if you wish to have your own frozen-counters with id: "id", you can add it via

```
#set-options(  
  frozen-counters: ("id": (real: . . , cover: . . ))  
)
```

and each time the counter step, you must put `#alias-counter("id").step(.,.)` or other updates so that it can correctly *rewind* your real counter.

4 List of Available Functions

4.1 Presentate Module

This is imported by default.

1. `#slide` -> content accepts:
 - `steps`: auto the number of subslides.
 - `body` the content.
2. `#set-options` -> state-update accepts:
 - `handout`: `false` handout mode, disabling the animations.
 - `drafted`: `false` drafted mode, placing the current subslide number on your slides.
 - `freeze-counter`: `true` whether to freeze the counters.

4.2 Animation Module

This is imported by default.

1. `#pause` -> content pause marker.
2. `#meanwhile` -> content reset the pause marker.
3. `#only` -> content accepts:
 - `..number` the subslide to show the content.
 - `body` the content
 - `hider`: it => none default cover function to hide the content.

4. `#uncover` -> content accepts the same arguments as `#only` but with default cover function being typst's hide function.
5. `#animate` -> dictionary accepts:
 - `cover`: hide the cover function used by `#uncover`.
 - `clear`: `it => none` the cover function used by `#only`.
 - `marker`: `it => it` a wrapper function that accepts a state update key (content type) and return it to the document.
 - `combine`: `(it, mark) => it + mark` another wrapper function that wraps the output of the cover functions (`it`)

and the marker (mark). This is useful when dealing with special input environment like CeTZ and Fletcher.

returns a dictionary containing only and uncover functions. **They are contextual**, so `#context {}` is needed when calling them.

6. `#reducer` -> content accepts:

- `cover`: hide the cover function to be used.
- `func`: (`..args`) => none the function to be reduced (i.e. its arguments being parsed).

4.3 Themes Module

Use with themes prefix, or import to your document. The demo.typ you are reading is created with default theme. Use it with

```
#import themes.default: *  
#show: template.with(  
  aspect-ratio: "16-9"  
)  
// your content goes here.
```

4.4 Store Module

Use with `store.` prefix.

- `#store.subslides` -> state stores the current subslide number.
- `#store.dynamics` -> state stores a dictionary containing
 - `pause`: int current number of pauses.
 - `steps`: int current minimum number of subslides needed to render all of the animations.

You can use this module to create your own animation functions, by get the subslide from `store.subslides` and update the new number of required steps to `store.dynamics`.