Presentate: Dynamic Features

A systematic test of animation tools

David Hajage | 2026-01-30

Introduction

Welcome

This presentation demonstrates the dynamic capabilities of **Presentate**.

We will cover:

Welcome

This presentation demonstrates the dynamic capabilities of **Presentate**.

We will cover:

- Basic flow control (`pause`, `fragments`)

Welcome

This presentation demonstrates the dynamic capabilities of **Presentate**.

We will cover:

- Basic flow control (`pause`, `fragments`)
- Precise visibility (`uncover`, `only`)

Welcome

This presentation demonstrates the dynamic capabilities of **Presentate**.

We will cover:

- Basic flow control (`pause`, `fragments`)
- Precise visibility (`uncover`, `only`)
- Relative indices and timeline synchronization

Welcome

This presentation demonstrates the dynamic capabilities of **Presentate**.

We will cover:

- Basic flow control (`pause`, `fragments`)
- Precise visibility (`uncover`, `only`)
- Relative indices and timeline synchronization
- Content transformations (`alert`, `transform`)

Welcome

This presentation demonstrates the dynamic capabilities of **Presentate**.

We will cover:

- Basic flow control (`pause`, `fragments`)
- Precise visibility (`uncover`, `only`)
- Relative indices and timeline synchronization
- Content transformations (`alert`, `transform`)
- Advanced package integration (`render`)

Basic Flow Control

Using #pause

The `#pause` function (or `#show: pause`) allows you to reveal content in chunks.

Using `#pause`

The `#pause` function (or `#show: pause`) allows you to reveal content in chunks.

Chunk 1: First, this line appears.

Using #pause

The `#pause` function (or `#show: pause`) allows you to reveal content in chunks.

Chunk 1: First, this line appears.

Chunk 2: Then, this second line is revealed.

Using #pause

The `#pause` function (or `#show: pause`) allows you to reveal content in chunks.

Chunk 1: First, this line appears.

Chunk 2: Then, this second line is revealed.

Chunk 3: Finally, you see this one.

Using #pause

The `#pause` function (or `#show: pause`) allows you to reveal content in chunks.

Chunk 1: First, this line appears.

Chunk 2: Then, this second line is revealed.

Chunk 3: Finally, you see this one.

You can even pause inside math:

$$(a + b)^2$$

Using #pause

The `#pause` function (or `#show: pause`) allows you to reveal content in chunks.

Chunk 1: First, this line appears.

Chunk 2: Then, this second line is revealed.

Chunk 3: Finally, you see this one.

You can even pause inside math:

$$(a + b)^2 = a^2 + 2ab + b^2$$

Using `#fragments`

`#fragments` is a shorthand for revealing multiple pieces of content one after another.

Using `#fragments`

`#fragments` is a shorthand for revealing multiple pieces of content one after another.

Fragment A: Individual content…

Using `#fragments`

`#fragments` is a shorthand for revealing multiple pieces of content one after another.

Fragment A: Individual content…Fragment B: …revealed one by one…

Using `#fragments`

`#fragments` is a shorthand for revealing multiple pieces of content one after another.

Fragment A: Individual content…Fragment B: …revealed one by one…Fragment C: … without multiple pause calls.

Using `#step-item`

`#step-item` is optimized for lists. It can even hide list markers and supports nesting.

Using `#step-item`

`#step-item` is optimized for lists. It can even hide list markers and supports nesting.

- Outer Item 1

Using `#step-item`

`#step-item` is optimized for lists. It can even hide list markers and supports nesting.

- Outer Item 1
  - Nested A

Using `#step-item`

`#step-item` is optimized for lists. It can even hide list markers and supports nesting.

- Outer Item 1
  - Nested A
  - Nested B

Using `#step-item`

`#step-item` is optimized for lists. It can even hide list markers and supports nesting.

- Outer Item 1
  - Nested A
  - Nested B
- Outer Item 2

Using `#step-item`

`#step-item` is optimized for lists. It can even hide list markers and supports nesting.

- Outer Item 1
  - Nested A
  - Nested B
- Outer Item 2
- Outer Item 3

Precise Visibility

`#uncover` vs `#only`

These functions allow you to show content on specific subslides.

uncover

- Preserves space when hidden.
- Uses `hide()` by default.

only

- Discards space when hidden.
- Content is completely removed.

`#uncover` vs `#only`

These functions allow you to show content on specific subslides.

- Preserves space when hidden.
- Uses `hide()` by default.

Shown on subslides 2 and 4.

- Discards space when hidden.
- Content is completely removed.

Shown on subslides 2 and 4.

`#uncover` vs `#only`

These functions allow you to show content on specific subslides.

- Preserves space when hidden.
- Uses `hide()` by default.

Shown from subslide 3 onwards.

- Discards space when hidden.
- Content is completely removed.

Shown from subslide 3 onwards.

`#uncover` vs `#only`

These functions allow you to show content on specific subslides.

- Preserves space when hidden.
- Uses `hide()` by default.

Shown on subslides 2 and 4. Shown from subslide 3 onwards.

- Discards space when hidden.
- Content is completely removed.

Shown on subslides 2 and 4. Shown from subslide 3 onwards.

Relative Indices

Relative Indices: `auto`, `none`, `rel`

Instead of hardcoding subslide numbers, use relative indices.

- Current pause state: Content A
- : Shown **after** the current pause (next step).
- : Shown **at** the same time as the current pause.
- : Shown 2 steps after the current pause.

Relative Indices: `auto`, `none`, `rel`

Instead of hardcoding subslide numbers, use relative indices.

- Current pause state: Content A Content B
- : Shown **after** the current pause (next step).
- `none`: Shown **at** the same time as the current pause.
- : Shown 2 steps after the current pause.

Relative Indices: `auto`, `none`, `rel`

Instead of hardcoding subslide numbers, use relative indices.

- Current pause state: Content A Content B
- `auto`: Shown **after** the current pause (next step).
- : Shown **at** the same time as the current pause.
- : Shown 2 steps after the current pause.

Relative Indices: `auto`, `none`, `rel`

Instead of hardcoding subslide numbers, use relative indices.

- Current pause state: Content A Content B
- : Shown **after** the current pause (next step).
- : Shown **at** the same time as the current pause.
- `(rel: 2)`: Shown 2 steps after the current pause.

Timeline Synchronization

Use `update-pause: true` to make subsequent pauses “aware” of the subslides added by `uncover` or `only`.

1. Regular Step
2. Hidden Step
3. This step waits for the surprise because of `update-pause`.

Timeline Synchronization

Use `update-pause: true` to make subsequent pauses “aware” of the subslides added by `uncover` or `only`.

1. Regular Step
2. Hidden Step
3. This step waits for the surprise because of `update-pause`.

Timeline Synchronization

Use `update-pause: true` to make subsequent pauses “aware” of the subslides added by `uncover` or `only`.

1. Regular Step
2. Hidden Step SURPRISE!
3. This step waits for the surprise because of `update-pause`.

Content Transformations

Alerts and Emphasis

`#alert` wraps content in a function (default is `emph`) on a specific subslide.

Alerts and Emphasis

`#alert` wraps content in a function (default is `emph`) on a specific subslide.

- Item 1

Alerts and Emphasis

`#alert` wraps content in a function (default is `emph`) on a specific subslide.

- Item 1
- Item 2 (Alerted!)

Alerts and Emphasis

`#alert` wraps content in a function (default is `emph`) on a specific subslide.

- Item 1
- *Item 2 (Alerted!)*
- Item 3

Alerts and Emphasis

`#alert` wraps content in a function (default is `emph`) on a specific subslide.

- Item 1
- Item 2 (Alerted!)
- Item 3

You can customize the alert function: DANGER ALERT

Alerts and Emphasis

`#alert` wraps content in a function (default is `emph`) on a specific subslide.

- Item 1
- Item 2 (Alerted!)
- Item 3

You can customize the alert function: DANGER ALERT

Complex Transformations

`#transform` allows a sequence of functions to be applied to content across subslides.

Complex Transformations

`#transform` allows a sequence of functions to be applied to content across subslides.

DYNAMISM

Complex Transformations

`#transform` allows a sequence of functions to be applied to content across subslides.

DYNAMISM

Complex Transformations

`#transform` allows a sequence of functions to be applied to content across subslides.

~~DYNAMISM~~

Advanced Integration

The `#render` Workspace

`#render` is the most powerful tool for integrating other packages (CeTZ, Fletcher). It lets you manually update the subslide state `s`.

Subslide 1: Cooling Down

The `#render` Workspace

`#render` is the most powerful tool for integrating other packages (CeTZ, Fletcher). It lets you manually update the subslide state `s`.

Subslide 2: Heating Up!

The `#render` Workspace

`#render` is the most powerful tool for integrating other packages (CeTZ, Fletcher). It lets you manually update the subslide state `s`.

Subslide 3: BOILING POINT!

Conclusion

Summary

Presentate provides a robust framework for:

Summary

Presentate provides a robust framework for:

- Chaining animations easily.

Summary

Presentate provides a robust framework for:

- Chaining animations easily.
- Targeting specific subslides.

Summary

Presentate provides a robust framework for:

- Chaining animations easily.
- Targeting specific subslides.
- Synchronizing multiple components.

Summary

Presentate provides a robust framework for:

- Chaining animations easily.
- Targeting specific subslides.
- Synchronizing multiple components.
- Hooking into external drawing libraries.

Summary

Presentate provides a robust framework for:

- Chaining animations easily.
- Targeting specific subslides.
- Synchronizing multiple components.
- Hooking into external drawing libraries.

Happy Presenting!