`Polylux`: Easily creating slides in Typst

An overview over all the features

Andreas Kröpelin

April 2023

Introduction

About this presentation

This presentation is supposed to briefly showcase what you can do with this package.

For a full documentation, read the online book.

A title

Let's explore what we have here.

On the top of this slide, you can see the slide title.

We used the `title` argument of the `#slide` function for that:

```
#slide(title: "First slide")[
    ...
]
```

(This works because we utilise the `clean` theme; more on that later.)

Titles are not mandatory, this slide doesn't have one.

But did you notice that the current section name is displayed above that top line?

We defined it using `#new-section-slide("Introduction")`.

This helps our audience with not getting lost after a microsleep.

You can also spot a short title above that.

The bottom of the slide

Now, look down!

There we have some general info for the audience about what talk they are actually attending right now.

You can also see the slide number there.

Dynamic content

A dynamic slide with pauses

Sometimes we don’t want to display everything at once.

A dynamic slide with pauses

Sometimes we don’t want to display everything at once.

That’s what the #pause function is there for!

A dynamic slide with pauses

Sometimes we don’t want to display everything at once.

That’s what the #pause function is there for!

It makes everything after it appear at the next subslide.

(Also note that the slide number does not change while we are here.)

Fine-grained control

When `#pause` does not suffice, you can use more advanced commands to show or hide content.

These are some of your options:

- `#uncover`
- `#only`
- `#alternatives`
- `#one-by-one`
- `#line-by-line`

Let's explore them in more detail!

`#uncover`: Reserving space

With `#uncover`, content still occupies space, even when it is not displayed.

For example, are only visible on the second “subslide”.

In `()` behind `#uncover`, you specify *when* to show the content, and in `[]` you then say *what* to show:

```
#uncover(3)[Only visible on the third "subslide"]
```

`#uncover`: Reserving space

With `#uncover`, content still occupies space, even when it is not displayed.

For example, these words are only visible on the second “subslide”.

In `()` behind `#uncover`, you specify *when* to show the content, and in `[]` you then say *what* to show:

```
#uncover(3)[Only visible on the third "subslide"]
```

`#uncover`: Reserving space

With `#uncover`, content still occupies space, even when it is not displayed.

For example, are only visible on the second “subslide”.

In `()` behind `#uncover`, you specify *when* to show the content, and in `[]` you then say *what* to show:

```
#uncover(3)[Only visible on the third "subslide"]
```

Only visible on the third “subslide"

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

Visible on subslides 1, 3, and 4

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Visible on subslides 2, 3, and 4

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

Visible on subslides 1, 3, and 4

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Visible on subslides 2, 3, and 4

Complex display rules

So far, we only used single subslide indices to define when to show something.

We can also use arrays of numbers...

```
#uncover((1, 3, 4))[Visible on subslides 1, 3, and 4]
```

Visible on subslides 1, 3, and 4

...or a dictionary with `beginning` and/or `until` keys:

```
#uncover((beginning: 2, until: 4))[Visible on subslides 2, 3, and 4]
```

Visible on subslides 2, 3, and 4

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Convenient rules as strings

As as short hand option, you can also specify rules as strings in a special syntax.

Comma separated, you can use rules of the form

`1-3`	from subslide 1 to 3 (inclusive)
`-4`	all the time until subslide 4 (inclusive)
`2-`	from subslide 2 onwards
`3`	only on subslide 3

```
#uncover("-2, 4-6, 8-")[Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards]
```

Visible on subslides 1, 2, 4, 5, 6, and from 8 onwards

`#only`: Reserving no space

Everything that works with `#uncover` also works with `#only`.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

`#only`: Reserving no space

Everything that works with `#uncover` also works with `#only`.

However, content is completely gone when it is not displayed.

For example, see how the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

`#only`: Reserving no space

Everything that works with `#uncover` also works with `#only`.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

`#only`: Reserving no space

Everything that works with `#uncover` also works with `#only`.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

`#only`: Reserving no space

Everything that works with `#uncover` also works with `#only`.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

`#only`: Reserving no space

Everything that works with `#uncover` also works with `#only`.

However, content is completely gone when it is not displayed.

For example, the rest of this sentence moves.

Again, you can use complex string rules, if you want.

```
#only("2-4, 6")[Visible on subslides 2, 3, 4, and 6]
```

Visible on subslides 2, 3, 4, and 6

`#alternatives`: Substituting content

You might be tempted to try

```
#only(1)[Ann] #only(2)[Bob] #only(3)[Christopher] likes #only(1)[chocolate] #only(2)[strawberry]
#only(3)[vanilla] ice cream.
```

Ann likes chocolate ice cream.

But it is hard to see what piece of text actually changes because everything moves around. Better:

```
#alternatives[Ann][Bob][Christopher] likes #alternatives[chocolate][strawberry][vanilla] ice cream.
```

Ann likes chocolate ice cream.

`#alternatives`: Substituting content

You might be tempted to try

```
#only(1)[Ann] #only(2)[Bob] #only(3)[Christopher] likes #only(1)[chocolate] #only(2)[strawberry]
#only(3)[vanilla] ice cream.
```

Bob likes strawberry ice cream.

But it is hard to see what piece of text actually changes because everything moves around.
Better:

```
#alternatives[Ann][Bob][Christopher] likes #alternatives[chocolate][strawberry][vanilla] ice cream.
```

Bob likes strawberry ice cream.

`#alternatives`: Substituting content

You might be tempted to try

```
#only(1)[Ann] #only(2)[Bob] #only(3)[Christopher] likes #only(1)[chocolate] #only(2)[strawberry]
#only(3)[vanilla] ice cream.
```

Christopher likes vanilla ice cream.

But it is hard to see what piece of text actually changes because everything moves around. Better:

```
#alternatives[Ann][Bob][Christopher] likes #alternatives[chocolate][strawberry][vanilla] ice cream.
```

Christopher likes vanilla ice cream.

`#one-by-one`: An alternative for `#pause`

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one

`#one-by-one`: An alternative for `#pause`

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

one

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one by

#one-by-one: An alternative for #pause

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

one by

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one by one

`#one-by-one`: An alternative for `#pause`

`#alternatives` is to `#only` what `#one-by-one` is to `#uncover`.

`#one-by-one` behaves similar to using `#pause` but you can additionally state when uncovering should start.

```
#one-by-one(start: 2)[one ][by ][one]
```

one by one

`start` can also be omitted, then it starts with the first subside:

```
#one-by-one[one ][by ][one]
```

one by one

`#line-by-line`: syntactic sugar for `#one-by-one`

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

`start` is again optional and defaults to `1`.

`#line-by-line`: syntactic sugar for `#one-by-one`

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

- first

`start` is again optional and defaults to `1`.

`#line-by-line`: syntactic sugar for `#one-by-one`

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

- first
- second

`start` is again optional and defaults to `1`.

`#line-by-line`: syntactic sugar for `#one-by-one`

Sometimes it is convenient to write the different contents to uncover one at a time in subsequent lines.

This comes in especially handy for bullet lists, enumerations, and term lists.

```
#line-by-line(start: 2)[
    - first
    - second
    - third
]
```

- first
- second
- third

`start` is again optional and defaults to `1`.

`#list-one-by-one` and Co: when `#line-by-line` doesn't suffice

While `#line-by-line` is very convenient syntax-wise, it fails to produce more sophisticated bullet lists, enumerations or term lists. For example, non-tight lists are out of reach.

For that reason, there are `#list-one-by-one`, `#enum-one-by-one`, and `#terms-one-by-one`, respectively.

```
#enum-one-by-one(start: 2, tight: false,
numbering: "i)")[first][second][third]
```

ii) first

iii) second

iv) third

Note that, for technical reasons, the bullet points, numbers, or terms are never covered.

`start` is again optional and defaults to `1`.

`#list-one-by-one` and Co: when `#line-by-line` doesn't suffice

While `#line-by-line` is very convenient syntax-wise, it fails to produce more sophisticated bullet lists, enumerations or term lists. For example, non-tight lists are out of reach.

For that reason, there are `#list-one-by-one`, `#enum-one-by-one`, and `#terms-one-by-one`, respectively.

```
#enum-one-by-one(start: 2, tight: false,
numbering: "i)")[first][second][third]
```

i) first

ii)

iii)

Note that, for technical reasons, the bullet points, numbers, or terms are never covered.

`start` is again optional and defaults to `1`.

`#list-one-by-one` and Co: when `#line-by-line` doesn't suffice

While `#line-by-line` is very convenient syntax-wise, it fails to produce more sophisticated bullet lists, enumerations or term lists. For example, non-tight lists are out of reach.

For that reason, there are `#list-one-by-one`, `#enum-one-by-one`, and `#terms-one-by-one`, respectively.

```
#enum-one-by-one(start: 2, tight: false,
numbering: "i)")[first][second][third]
```

i) first

ii) second

iii)

Note that, for technical reasons, the bullet points, numbers, or terms are never covered.

`start` is again optional and defaults to `1`.

`#list-one-by-one` and Co: when `#line-by-line` doesn't suffice

While `#line-by-line` is very convenient syntax-wise, it fails to produce more sophisticated bullet lists, enumerations or term lists. For example, non-tight lists are out of reach.

For that reason, there are `#list-one-by-one`, `#enum-one-by-one`, and `#terms-one-by-one`, respectively.

```
#enum-one-by-one(start: 2, tight: false,
numbering: "i)")[first][second][third]
```

i) first

ii) second

iii) third

Note that, for technical reasons, the bullet points, numbers, or terms are never covered.

`start` is again optional and defaults to `1`.

Themes

How a slide looks…

… is defined by the *theme* of the presentation.

This demo uses the `clean` theme.

Because of it, the title slide and the decoration on each slide (with section name, short title, slide number etc.) look the way they do.

Themes can also provide variants, for example …

... this one!

It’s very minimalist and helps the audience focus on an important point.

Your own theme?

If you want to create your own design for slides, you can define custom themes!

The book explains how to do so.

Utilities

The `utils` module

Polylux ships a `utils` module with solutions for common tasks in slide building.

Fit to height

You can scale content such that it has a certain height using `#fit-to-height(height, content)`:

Height is 2.5cm

Fill remaining space

This function also allows you to fill the remaining space by using fractions as heights, i.e. `fit-to-height(1fr)[...]`:

Wow!

Side by side content

Often you want to put different content next to each other. We have the function `#side-by-side` for that:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.

Outline

Why not include an outline?

Typst features

Use Typst!

Typst gives us so many cool things[1]. Use them!

[1] For example footnotes!

Bibliography

Let us cite something so we can have a bibliography: [1] [2] [3]

[1] “A.”

[2] “B.”

[3] “C.”

Conclusion

That's it!

Hopefully you now have some kind of idea what you can do with this template.

Consider giving it a GitHub star ☆ or open an issue if you run into bugs or have feature requests.