Definition 1. This is a package for theorem environments.

theofig

figure implementation of theorem environments

A Typst package for creation and customization
of theorem environments built on top of `std.figure`.

`github.com/Danila-Bain/typst-theorems`

Version 0.0.1

Guide

Reference

Usage examples

Importing everything with * is recommended:

```
#import "@preview/theofig:0.0.1": *
```

Basic usage

```
#theorem[#lorem(5)] <theorem-1>

#theorem[Lorem][#lorem(10)]

#proof[It follows from @theorem-1.]
```

Theorem 1. Lorem ipsum dolor sit amet.

Theorem 2 (Lorem). Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Proof. It follows from Theorem 1. ∎

,

Default environments

`theofig` defines a number of default environments with sensible default style.

```
#theorem[#lorem(5)]

#lemma[#lorem(5)]

#statement[#lorem(5)]

#remark[#lorem(5)]

#corollary[#lorem(5)]

#example[#lorem(5)]

#definition[#lorem(5)]

#algorithm[#lorem(5)]

#proof[#lorem(5)]

#problem[#lorem(5)]

#solution[#lorem(5)]
```

Theorem 1. Lorem ipsum dolor sit amet.

Lemma 1. Lorem ipsum dolor sit amet.

Statement 1. Lorem ipsum dolor sit amet.

Remark 1. Lorem ipsum dolor sit amet.

Corollary. Lorem ipsum dolor sit amet.

Example 1. Lorem ipsum dolor sit amet.

Definition 1. Lorem ipsum dolor sit amet.

Algorithm 1. Lorem ipsum dolor sit amet.

Proof. Lorem ipsum dolor sit amet. ∎

Problem 1. Lorem ipsum dolor sit amet.

Solution. Lorem ipsum dolor sit amet.

Custom environments

All default environments of `theofig` are defined as `with`-specializations of a function `theofig.theofig`, which can be used to create custom environments.

```
#let joke = theofig.with(supplement: "Joke")

#{theofig-kinds += ("joke", )}

#show figure-where-kind-in(theofig-kinds): set
text(gradient.linear(red, green, blue))

#joke[Why was six afraid of seven? Because 7 8 9.]
#joke[
  Parallel lines have so much in common...
  It's a shame they'll never meet.
]
#statement[A topologist is someone who can't tell
the difference between a coffee mug and a
doughnut.]
```

Joke 1. Why was six afraid of seven? Because 7 8 9.

Joke 2. Parallel lines have so much in common... It's a shame they'll never meet.

Statement 1. A topologist is someone who can't tell the difference between a coffee mug and a doughnut.

If supplement is specified, the `kind` of figure is chosen automatically as `lower(supplement)`, so in this example adding `"joke"` to `theofig-kinds` lets us apply styling to a `joke` environment together with all standard environments.

Languages support

Enabled by default argument `translate-supplement: true`, `theofig` environments map supplement based on context-dependant `text.lang`. List of supported languages: en, ru, de, fr, es, it, pt, pl, cs, zh, ja, ko, ar. Note that unlike supplement of a figure, a supplement in reference changes if `text.lang` is not the same as it was at the location of the figure.

```
#set text(lang: "ru")
#theorem[#lorem(5)]

#set text(lang: "es")
#theorem[#lorem(5)]

#set text(lang: "de")
#theorem[#lorem(5)]

#set text(lang: "ja")
#theorem[#lorem(5)]
```

Теорема 1. Lorem ipsum dolor sit amet.

Teorema 2. Lorem ipsum dolor sit amet.

Satz 3. Lorem ipsum dolor sit amet.

定理 4. Lorem ipsum dolor sit amet.

Ways to specify numbering style

Most default environments use `figure`'s default numbering, which is `"1"` (see Definition 1). Hence, we can change default numbering for many environments simultaneously using a `show` rule with `set figure(numbering: ...)` (see Definition II). Then, if we specify an argument `numbering` in the environment, it takes priority over `figure`'s numbering (see Definition C and Definition (iv)).

```
#definition[Default @def-a-1.]<def-a-1>

#show figure-where-kind-in(
  theofig-kinds
): set figure(numbering: "I")
#definition[Show rule @def-a-2.]<def-a-2>

#let definition = definition.with(numbering: "A")
#definition[Redefined @def-a-3.]<def-a-3>

#definition(numbering: numbering.with("(i)"))[
  Argument @def-a-4.
]<def-a-4>
```

Definition 1. Default Definition 1.

Definition II. Show rule Definition II.

Definition C. Redefined Definition C.

Definition (iv). Argument Definition (iv).

Numbers out of order

Using the `theofig`'s argument `number`, we can specify a number regardless of automatic numeration. Passing a `label` to `number` copies the numbering of the same environment by that `label`, which is useful for alternative definitions or equivalent statements of theorems (see Definition 2 and Definition 2').

```
#definition[Default.]

#definition(numbering: none)[No numbering.]

#definition[Equivalent to @def-2.]<def-1>

#definition(number: <def-1>, numbering: "1'")[
  Equivalent to @def-1.
]<def-2>

#definition(number: 100)[
  This is @def-100.
]<def-100>

#definition(number: 5, numbering: "A")[
  This is @def-3.
]<def-3>

#definition(number: $e^pi$)[
  This is @def-exp
]<def-exp>

#definition[Back to default.]
```

Definition 1. Default.

Definition. No numbering.

Definition 2. Equivalent to Definition 2'.

Definition 2'. Equivalent to Definition 2.

Definition 100. This is Definition 100.

Definition E. This is Definition E.

Definition e^{π}. This is Definition e^{π}

Definition 3. Back to default.

Another use case for `number` argument is local numbering, such as multiple corollaries immediately after a theorem:

```
#theorem[]
#corollary[]
#theorem[]
#corollary(number: "1")[]
#corollary(number: "2")[]
#theorem[]
#corollary(number: "1")[]
#corollary(number: "2")[]
```

Theorem 1.

Corollary.

Theorem 2.

Corollary 1.

Corollary 2.

Theorem 3.

Corollary 1.

Corollary 2.

Shared numbering

If you want different environments to share numbering, you just need to have them have the same kind, but different supplement:

```
#let lemma     = lemma.with(kind: "theorem")
#let statement = statement.with(kind: "theorem")

#theorem[#lorem(5)]
#lemma[#lorem(5)]
#statement[#lorem(5)]
#theorem[#lorem(5)]
```

Theorem 1. Lorem ipsum dolor sit amet.

Lemma 2. Lorem ipsum dolor sit amet.

Statement 3. Lorem ipsum dolor sit amet.

Theorem 4. Lorem ipsum dolor sit amet.

One obvious limitation of that approach is that not only numbering will be shared. All styling of `theorem` that is based on `show` rules will also apply to `lemma` and `statement`. To mitigate that, styling can be applied individually through setting arguments `format-caption`, `format-body`, `block-options`, and `figure-options`.

```
#let theorem = theorem.with(
  format-body: emph,
)
#let lemma     = lemma.with(
  kind: "theorem",
  format-caption: none,
)
#let statement = statement.with(
  kind: "theorem",
  block-options: (
    stroke: 1pt, radius: 3pt, inset: 5pt,
  ),
)

#theorem[#lorem(5)]
#lemma[#lorem(5)]
#lemma[#lorem(5)]
```

Theorem 1. *Lorem ipsum dolor sit amet.*

Lemma 2. Lorem ipsum dolor sit amet.

Lemma 3. Lorem ipsum dolor sit amet.

Show rules to specify a style

All environments are `figure`'s under the hood, and they can be styled using show rules. The title of environment, such as "**Theorem 4 (Cauchy).**" can be style using `show figure.caption: ...` rules.

```
// apply to one
#show figure.where(kind: "theorem"): smallcaps
// apply to some
#show figure-where-kind-in(
  ("solution", "problem")
): emph
// apply to all
#show figure-where-kind-in(theofig-kinds): set
figure(
  numbering: "I",
)
// apply to all except some
#show figure-where-kind-in(
  theofig-kinds, except: ("proof",),
): set text(blue)

#definition[#lorem(10)]
#theorem[#lorem(10)]
#proof[#lorem(10)]
#problem[#lorem(10)]
#solution[#lorem(10)]
```

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

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

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

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

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

Style examples

```
#theorem[Default. #lorem(16)]

#show figure.where(kind: "definition"): it => {
  show figure.caption: emph
  show figure.caption: strong.with(delta: -300)
  it
}
#definition[Italic caption. #lorem(16)]

#show figure.where(kind: "lemma"): it => {
  show figure.caption: underline.with(offset:
1.5pt)
  show figure.caption: strong.with(delta: -300)
  it
}
#lemma[Underline caption. #lorem(16)]

#show figure.where(kind: "proposition"): it => {
  show: emph
  show figure.caption: emph
  show figure.caption: smallcaps
  show figure.caption: strong.with(delta: -300)
  it
}
#proposition[Italic body, smallcaps caption.
#lorem(12)]


#show figure.where(kind: "corollary"): it => {
  show figure.caption: strong.with(delta: -300)
  show figure.caption: set text(tracking: 3pt)
  it
}
#corollary[Sparse caption. #lorem(16)]

#show figure.where(kind: "statement"): block.with(
  stroke: 1pt, radius: 3pt, inset: 5pt,
)
#statement[Block. #lorem(16)]

#show figure.where(kind: "solution"): block.with(
  stroke: (left: 1pt), inset: (right: 0pt, rest:
5pt)
)
#solution[Line to the left. #lorem(16)]
```

Theorem 1. Default. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et.

Definition 1. Italic caption. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et.

Lemma 1. Underline caption. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et.

Proposition 1. *Italic body, smallcaps caption. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor.*

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

Statement 1. Block. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et.

Solution. Line to the left. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et.

Limitations

Because environments in `theofig` are implemented as figures, show rules applied to them affect nested figures of any kind, including images and tables:

```
#show figure.where(kind: "example"): it => {
  set figure(numbering: "I")
  show figure.caption: smallcaps
  it
}
#example[
  Example with an image:
  #figure(caption: [Example Image])[
    #image(
      bytes(range(256).map(i => i.bit-and(i*i))),
      format:(encoding:"luma8",width:32,height:8),
      width: 100%,
    )
  ]
]
```

Example I. Example with an image:

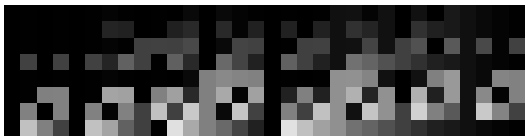

Figure I: Example Image

In this example, `smallcaps` is applied not only to the `example` title, but also to the actual `figure`'s caption, the same is true for numbering. It is an undesirable limitation, which leads us to either moving our figures outside of `theofig` environments or styling each `theofig` environment individually like in the following example.

```
#let example = example.with(
  numbering: "I",
  format-caption: it => smallcaps(strong(it)),
)
#example[
  Example with an image:
  #figure(caption: [Example Image])[
    #image(
      bytes(range(256).map(i => i.bit-and(i*i))),
      format:(encoding:"luma8",width:32,height:8),
      width: 100%,
    )
  ]
]
```

Example I. Example with an image:

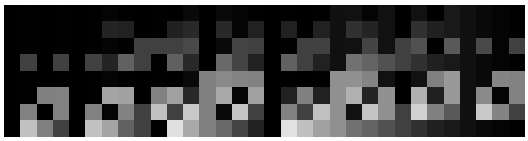

Figure 2: Example Image

Main functions

show-figure-caption()

Shorthand for

```
#{
  it => {
    show figure.caption: function1
    show figure.caption: function2
    ...
    it
  }
}
```

for a provided list of functions.

Example

```
#show figure.where(
  kind: "theorem"
): show-figure-caption(
  emph, strong.with(delta: -300)
)

#lemma[Default style.]
#theorem[Custom style.]
```

Lemma 1. Default style.

Theorem 1. Custom style.

If no functions are provided, show-figure-caption() is an identity function that leaves its argument unaffected.

show-figure-caption(..functions: array of functions)

..functions array of functions

List of functions that are applied to figure.caption by show figure.caption: func.

figure-where-kind-in()

Selector for firgures with selected kinds, combined with selector.or().

Use case: Apply a single style to many theorem-like environments in one #show rule, or to style all default theofig kinds while excluding some (e.g. exclude "proof"):

```
#show figure-where-kind-in(
  theofig-kinds, except: ("proof",)
): block.with(
    stroke: 1pt, radius: 3pt, inset: 5pt,
)
#definition[]
#theorem[]
#proof[]
```

Definition 1.

Theorem 1.

Proof. ∎

figure-where-kind-in(kinds: array, except: array)

kinds array

A list of figure kinds (e.g. "theorem", "definition", ...) to include in the selector.

except `array` default `()`

A list of figure kinds to exclude from the resulting selector.

`theofig-reset-counters()`

For every `kind` in `kinds` that is not in `except`, perform:

```
#counter(figure.where(kind: kind)).update(0)
```

which sets that figure counter to `0` (so next created figure increments from 1). Useful to restart numbering for groups of environments (e.g., at the start of a chapter, or before a block of examples).

Example

```
#let corollary = corollary.with(numbering: "1")
#show figure.where(kind: "theorem"): it => {
  theofig-reset-counters(("corollary",))
  it
}

#theorem[]
#corollary[Follows Theorem 1.]
#corollary[Follows Theorem 1.]

#theorem[]
#corollary[Follows Theorem 2.]
#corollary[Follows Theorem 2.]
```

Theorem 1.

Corollary 1. Follows Theorem 1.

Corollary 2. Follows Theorem 1.

Theorem 2.

Corollary 1. Follows Theorem 2.

Corollary 2. Follows Theorem 2.

```
theofig-reset-counters(kinds: array, except: array)
```

kinds `array`

List of figure kinds whose counters are to reset.

except `array` default `()`

List of kinds to skip when resetting.