

# typst-theorems

sahasatvik

<https://github.com/sahasatvik/typst-theorems>

## Contents

1. Introduction .....	1
2. Using typst-theorems .....	1
3. Feature demonstration .....	2
3.1. Proofs .....	3
3.2. Suppressing numbering .....	4
3.3. Limiting depth .....	4
3.4. Custom formatting .....	5
3.5. Labels and references .....	6
3.6. Overriding base .....	7
4. Function reference .....	7
4.1. thmenv .....	7
4.2. thmbox and thmplain .....	8
4.3. thmproof, proof-bodyfmt and qedhere .....	9
4.4. thmrules .....	9
5. Acknowledgements .....	9

## 1. Introduction

The typst-theorems package provides Typst functions that help create numbered theorem environments. This is heavily inspired by the `\newtheorem` functionality of LaTeX.

A *theorem environment* lets you wrap content together with automatically updating *numbering* information. Such environments use internal state counters for this purpose. Environments can

- share the same counter (*Theorems* and *Lemmas* often do so)
- keep a global count, or be attached to
  - other environments (*Corollaries* are often numbered based upon the parent *Theorem*)
  - headings
- have a numbering level depth fixed (for instance, use only top level heading numbers)
- be referenced elsewhere in the document, via `labels`

## 2. Using typst-theorems

Import all functions provided by typst-theorems using

```
#import "theorems.typ": *  
#show: thmrules
```

The second line is crucial for displaying thmenvs and references correctly!

The core of this module consists of `thmenv`. The functions `thmbox`, `thmplain`, and `thmproof` provide some simple defaults for the appearance of thmenvs.

### 3. Feature demonstration

Create box-like *theorem environments* using `thmbox`, a wrapper around `thmenv` which provides some simple defaults.

```
#let theorem = thmbox(  
  "theorem",           // identifier  
  "Theorem",           // head  
  fill: rgb("#e8e8f8")  
)
```

Such definitions are convenient to place in the preamble or a template; use the environment in your document via

```
#theorem("Euclid")[  
  There are infinitely many primes.  
] <euclid>
```

This produces the following.

**Theorem 3.1** (Euclid): There are infinitely many primes.

Note that the name is optional. This theorem environment will be numbered based on its parent heading counter, with successive theorems automatically updating the final index.

The `<euclid>` label can be used to refer to this Theorem via the reference `@euclid`. Go to Section 3.5 to read more.

You can create another environment which uses the same counter, say for *Lemmas*, as follows.

```
#let lemma = thmbox(  
  "theorem",           // identifier - same as that of theorem  
  "Lemma",             // head  
  fill: rgb("#efe6ff")  
)  
  
#lemma[  
  If  $n$  divides both  $x$  and  $y$ , it also divides  $x - y$ .  
]
```

**Lemma 3.2:** If  $n$  divides both  $x$  and  $y$ , it also divides  $x - y$ .

You can *attach* other environments to ones defined earlier. For instance, *Corollaries* can be created as follows.

```
#let corollary = thmbox(  
  "corollary",         // identifier  
  "Corollary",         // head  
  base: "theorem",     // base - use the theorem counter  
  fill: rgb("#f8e8e8")  
)  
  
#corollary(numbering: "1.1")[
```

```

    If  $n$  divides two consecutive natural numbers, then  $n = 1$ .
]

```

**Corollary 3.2.1:** If  $n$  divides two consecutive natural numbers, then  $n = 1$ .

Note that we have provided a numbering string; this can be any valid numbering pattern as described in the [numbering](#) documentation.

### 3.1. Proofs

The `thmproof` function gives nicer defaults for formatting proofs.

```

#let proof = thmproof("proof", "Proof")

#proof([of @euclid])[
  Suppose to the contrary that  $p_1, p_2, \dots, p_n$  is a finite enumeration
  of all primes. Set  $P = p_1 p_2 \dots p_n$ . Since  $P + 1$  is not in our list,
  it cannot be prime. Thus, some prime factor  $p_j$  divides  $P + 1$ . Since
   $p_j$  also divides  $P$ , it must divide the difference  $(P + 1) - P = 1$ , a
  contradiction.
]

```

*Proof of [Theorem 3.1](#):* Suppose to the contrary that  $p_1, p_2, \dots, p_n$  is a finite enumeration of all primes. Set  $P = p_1 p_2 \dots p_n$ . Since  $P + 1$  is not in our list, it cannot be prime. Thus, some prime factor  $p_j$  divides  $P + 1$ . Since  $p_j$  also divides  $P$ , it must divide the difference  $(P + 1) - P = 1$ , a contradiction. ■

If your proof ends in a block equation, or a list/enum, you can place `qedhere` to correctly position the `qed` symbol.

```

#theorem[
  There are arbitrarily long stretches of composite numbers.
]
#proof[
  For any  $n > 2$ , consider  $\begin{aligned}
& n! + 2, \quad n! + 3, \quad \dots, \quad n! + n \text{ #qedhere} \\
& \$
\end{aligned}$ 
]

```

**Theorem 3.1.1:** There are arbitrarily long stretches of composite numbers.

*Proof:* For any  $n > 2$ , consider

$$n! + 2, \quad n! + 3, \quad \dots, \quad n! + n$$

**Caution:** The `qedhere` symbol does not play well with numbered/multiline equations!

You can set a custom `qed` symbol (say  $\square$ ) by setting the appropriate option in `thmrules` as follows.

```

#show: thmrules.with(qed-symbol:  $\square$ )

```

## 3.2. Suppressing numbering

Supplying `numbering: none` to an environment suppresses numbering for that block, and prevents it from updating its counter.

```
#let example = thmplain(  
  "example",  
  "Example"  
) .with(numbering: none)  
  
#example[  
  The numbers $2$, $3$, and $17$ are prime.  
]
```

*Example:* The numbers 2, 3, and 17 are prime.

Here, we have used the `thmplain` function, which is identical to `thmbox` but sets some plainer defaults. You can also write

```
#lemma(numbering: none)[  
  The square of any even number is divisible by $4$.  
]  
#lemma[  
  The square of any odd number is one more than a multiple of $4$.  
]
```

**Lemma:** The square of any even number is divisible by 4.

**Lemma 3.2.1:** The square of any odd number is one more than a multiple of 4.

Note that the last *Lemma* is *not* numbered 3.1.2!

You can also override the automatic numbering as follows.

```
#lemma(number: "42")[  
  The square of any natural number cannot be two more than a multiple of 4.  
]
```

**Lemma 42:** The square of any natural number cannot be two more than a multiple of 4.

Note that this does *not* affect the counters either!

## 3.3. Limiting depth

You can limit the number of levels of the base numbering used as follows.

```
#let definition = thmbox(  
  "definition",  
  "Definition",  
  base_level: 1,           // take only the first level from the base  
  stroke: rgb("#68ff68") + 1pt  
)
```

```
#definition("Prime numbers")[
  A natural number is called a _prime number_ if it is greater than $1$ and
  cannot be written as the product of two smaller natural numbers. <prime>
]
```

**Definition 3.1** (Prime numbers): A natural number is called a *prime number* if it is greater than 1 and cannot be written as the product of two smaller natural numbers.

Note that this environment is *not* numbered 3.2.1!

```
#definition("Composite numbers")[
  A natural number is called a _composite number_ if it is greater than $1$
  and not prime.
]
```

**Definition 3.2** (Composite numbers): A natural number is called a *composite number* if it is greater than 1 and not prime.

Setting a `base_level` higher than what `base` provides will introduce padded zeroes.

```
#example(base_level: 4, numbering: "1.1")[
  The numbers $4$, $6$, and $42$ are composite.
]
```

*Example 3.3.0.0.1:* The numbers 4, 6, and 42 are composite.

### 3.4. Custom formatting

The `thmbox` function lets you specify rules for formatting the title, the name, and the body individually. Here, the title refers to the head and number together.

```
#let proof-custom = thmplain(
  "proof",
  "Proof",
  base: "theorem",
  titlefmt: smallcaps,
  bodyfmt: body => [
    #body #h(1fr) $square$ // float a QED symbol to the right
  ]
).with(numbering: none)

#lemma[
  All even natural numbers greater than 2 are composite.
]
#proof-custom[
  Every even natural number $n$ can be written as the product of the natural
  numbers $2$ and $n/2$. When $n > 2$, both of these are smaller than $2$
  itself.
]
```

**Lemma 3.4.1:** All even natural numbers greater than 2 are composite.

PROOF: Every even natural number  $n$  can be written as the product of the natural numbers 2 and  $n/2$ . When  $n > 2$ , both of these are smaller than 2 itself.  $\square$

You can go even further and use the `thmenv` function directly. It accepts an identifier, a base, a `base_level`, and a `fmt` function.

```
#let notation = thmenv(
  "notation",           // identifier
  none,                 // base - do not attach, count globally
  none,                 // base_level - use the base as-is
  (name, number, body, color: black) => [
    // fmt - format content using the environment
    // name, number, body, and an optional color
    #text(color)[#h(1.2em) *Notation (#number) #name*]:
    #h(0.2em)
    #body
    #v(0.5em)
  ]
).with(numbering: "I") // use Roman numerals

#notation[
  The variable  $p$  is reserved for prime numbers.
]
#notation("for Reals", color: green)[
  The variable  $x$  is reserved for real numbers.
]
```

**Notation (I):** The variable  $p$  is reserved for prime numbers.

**Notation (II) for Reals:** The variable  $x$  is reserved for real numbers.

Note that the `color: green` named argument supplied to the theorem environment gets passed to the `fmt` function. In general, all extra named arguments supplied to the theorem will be passed to `fmt`. On the other hand, the positional argument "for Reals" will always be interpreted as the name argument in `fmt`.

```
#lemma(title: "Lem.", stroke: 1pt)[
  All multiples of 3 greater than 3 are composite.
]
```

**Lem. 3.4.2:** All multiples of 3 greater than 3 are composite.

Here, we override the `title` (which defaults to the head) as well as the `stroke` in the `fmt` produced by `thmbox`. All block arguments can be overridden in `thmbox` environments in this way.

### 3.5. Labels and references

You can place a `<label>` outside a theorem environment, and reference it later via `@` references! For example, go back to [Theorem 3.1](#).

Recall that there are infinitely many prime numbers via `@euclid`.

Recall that there are infinitely many prime numbers via [Theorem 3.1](#).

You can reference future environments too, like `@oddprime[Cor.]`.

You can reference future environments too, like [Cor. 3.5.1.1](#).

```
#lemma(supplement: "Lem.", refnumbering: "(1.1)")[
  All primes apart from $2$ and $3$ are of the form $6k$ plus.minus 1$.
] <primeform>
```

You can modify the supplement and numbering to be used in references, like `@primeform`.

**Lemma 3.5.1:** All primes apart from 2 and 3 are of the form  $6k \pm 1$ .

You can modify the supplement and numbering to be used in references, like [Lem. \(3.5.1\)](#).

**Caution:** Links created by references to thmenvs will be styled according to `#show link: rules`.

### 3.6. Overriding base

```
#let remark = thmplain("remark", "Remark", base: "heading")
#remark[
  There are infinitely many composite numbers.
]

#corollary[
  All primes greater than $2$ are odd.
] <oddprime>
#remark(base: "corollary")[
  Two is a _lone prime_.
]
```

*Remark 3.6.1:* There are infinitely many composite numbers.

**Corollary 3.5.1.1:** All primes greater than 2 are odd.

*Remark 3.5.1.1.1:* Two is a *lone prime*.

This remark environment, which would normally be attached to the current *heading*, now uses the corollary as a base.

## 4. Function reference

### 4.1. thmenv

The `thmenv` function produces a *theorem environment*.

```
#let thmenv(
  identifier,          // environment counter name
  base,                // base counter name, can be "heading" or none
  base_level,          // number of base number levels to use
  fmt                  // formatting function of the form
                      // (name, number, body, ..args) -> content
) = { ... }
```

The `fmt` function must accept a theorem name, number, body, and produce formatted content. It may also accept additional positional arguments, via `args`.

A *theorem environment* is itself a map of the following form.

```
(
  ..args,
  body,                // body content
  number: auto,         // number, overrides numbering if present
  numbering: "1.1",     // numbering style, can be a function
  refnumbering: auto,   // numbering style used in references,
                      // defaults to "numbering"
  supplement: identifier, // supplement used in references
  base: base,           // base counter name override
  base_level: base_level // base_level override
) -> content
```

Positional arguments in `args` are as follows

- `name`: The name of the theorem, typically displayed after the title.

All additional named arguments in `args` will be passed on to the associated `fmt` function supplied in `thmenv`.

## 4.2. `thmbox` and `thmplain`

The `thmbox` wraps `thmenv`, supplying a box-like `fmt` function.

```
#let thmbox(
  identifier,          // identifier
  head,                // head - common name, used in the title
  ..blockargs,         // named arguments, passed to #block
  supplement: auto,     // supplement for references, defaults to "head"
  padding: (top: 0.5em, bottom: 0.5em),
                      // box padding, passed to #pad
  namefmt: x => [(#x)], // formatting for name
  titlefmt: strong,     // formatting for title (head + number)
  bodyfmt: x => x,       // formatting for body
  separator: [#h(0.1em):#h(0.2em)],
                      // separator inserted between name and body
  base: "heading",      // base - defaults to using headings
  base_level: none,     // base_level - defaults to using base as-is
) = { ... }
```

The `thmbox` function sets the following defaults for the block.

```
(
  width: 100%,
  inset: 1.2em,
  radius: 0.3em,
  breakable: false,
)
```



The `thmplain` function is identical to `thmbox`, except with plainer defaults.

```
#let thmplain = thmbox.with(  
  padding: (top: 0em, bottom: 0em),  
  breakable: true,  
  inset: (top: 0em, left: 1.2em, right: 1.2em),  
  namefmt: name => emph([(#name)]),  
  titlefmt: emph,  
)
```

### 4.3. `thmproof`, `proof-bodyfmt` and `qedhere`

The `thmproof` function is identical to `thmplain`, except with defaults appropriate for proofs.

```
#let thmproof(..args) = thmplain(  
  ..args,  
  namefmt: emph,  
  bodyfmt: proof-bodyfmt,  
  ..args.named()  
) .with(numbering: none)
```

The `proof-bodyfmt` function is a `bodyfmt` function that automatically places a `qed` symbol at the end of the body.

You can place `#qedhere` inside a block equation, or at the end of a list/enum item to place the `qed` symbol on the same line.

### 4.4. `thmrules`

The `thmrules` show rule sets important styling rules for theorem environments, references, and equations in proofs.

```
#let thmrules(  
  qed-symbol: $qed$,      // QED symbol used in proofs  
  doc  
) = { ... }
```

## 5. Acknowledgements

Thanks to

- [MJHutchinson](#) for suggesting and implementing the `base_level` and `base: none` features,
- [rmolinari](#) for suggesting and implementing the `separator: ...` feature,
- [DVDTSB](#) for contributing
  - the idea of passing named arguments from the theorem directly to the `fmt` function.
  - the `number: ...` override feature.
  - the `title: ...` override feature in `thmbox`.
- The awesome devs of [typst.app](#) for their support.