

# Scripst Documentation

## Article Style Set

AnZrew

AnZreww

AnZrewww

2025-02-28

**Abstract:** Scripst is a simple and easy-to-use Typst language template, suitable for various scenarios such as daily documents, assignments, notes, papers, etc.

**Keywords:** Scripst, Typst, template

## Contents

<b>1</b>	<b>Typesetting Typst Documents with Scripst</b>	<b>2</b>
1.1	Using Typst	2
1.2	Using Scripst	3
1.2.1	Using Extracted Files	3
1.2.2	Local Package Management	4
1.2.3	Online Package Management	4
<b>2</b>	<b>Template Parameter Description</b>	<b>5</b>
2.1	template	5
2.2	title	6
2.3	info	6
2.4	author	6
2.5	time	6
2.6	abstract	7
2.7	keywords	7
2.8	font_size	7
2.9	contents	8
2.10	content_depth	8
2.11	matheq_depth	8
2.12	lang	8
2.13	body	8
<b>3</b>	<b>Template Effect Display</b>	<b>9</b>

3.1	Front Page	9
3.2	Table of Contents	9
3.3	Fonts and Environments	9
3.3.1	Fonts	9
3.3.2	Environments	9
3.3.3	Lists	11
3.3.4	Quotes	11
3.3.5	Links	11
3.3.6	Hyperlinks and Citations	12
3.4	<code>#newpara()</code> Function	12
3.5	<code>countblock</code>	12
3.5.1	Creating and Registering <code>countblock</code>	12
3.5.2	Using <code>countblock</code>	13
<b>4</b>	<b>Conclusion</b>	<b>15</b>

Typst is a simple document generation language with syntax similar to lightweight Markdown markup. Using appropriate `set` and `show` commands, you can highly customise the style of your documents.

Scripst is a simple and easy-to-use Typst language template, suitable for various scenarios such as daily documents, assignments, notes, papers, etc.

## I Typesetting Typst Documents with Scripst

### 1.1 Using Typst

Typst is a lighter language to use compared to LaTeX. Once the template is written, you can complete the document writing with lightweight markup similar to Markdown.

Compared to LaTeX, Typst has the following advantages:

- Extremely fast compilation speed
- Simple and lightweight syntax
- Strong code extensibility
- Easier mathematical formula input
- ...

Therefore, Typst is very suitable for writing lightweight daily documents. You can get even better typesetting results than LaTeX with the time cost of writing Markdown.

You can install Typst in the following ways:

```
sudo apt install typst # Debian/Ubuntu
sudo pacman -S typst # Arch Linux
winget install --id Typst.Typst # Windows
brew install typst # macOS
```

You can also find more information in the [Typst GitHub repository](#).


## 1.2 Using Scripst

Based on Typst, Scripst provides some simple templates for convenient daily document generation.

### 1.2.1 Using Extracted Files

You can find and download the Scripst templates in the [Scripst GitHub repository](#).

You can choose <> **code** → **Download ZIP** to download the Scripst templates. When using them, just place the template files in your document directory and import the template files at the beginning of your document.

 Consider the project directory structure to correctly import the template files.

```
project/
├── src/
│   ├── main.typ
│   ├── ...
│   └── components.typ
├── pic/
│   └── ...
├── main.typ
├── chap1.typ
├── chap2.typ
└── ...
```

If the project directory structure is as shown above, then the way to import the template files in `main.typ` should be:

```
#import "src/main.typ": *
```

The advantage of this method is that you can adjust some parameters in the template at any time. Since the template is designed modularly, you can easily find and modify the parts you need to change.

### 1.2.2 Local Package Management

A better way is to refer to the official [local package management documentation](#) and place the template files in the local package management directory `{data-dir}/typst/packages/{namespace}/{name}/{version}`, so you can use the Scripst templates anywhere.

Of course, you don't have to worry about not being able to modify the template files. You can directly use `#set`, `#show` commands in the document to override some parameters in the template.

For example, the template should be placed in

```
~/.local/share/typst/packages/local/scripst/1.1.0 # in Linux
%APPDATA%\typst\packages\local\scripst\1.1.0      # in Windows
```

If the directory structure is like this, then the way to import the template files in the document should be:

```
#import "@local/scripst:1.1.0": *
```

The advantage of this is that you can directly use `typst init` to create a new project with the template:

```
typst init @local/scripst:1.1.0 project_name
```

### 1.2.3 Online Package Management

We will submit it to the community as soon as possible so that you can directly use

```
#import "@preview/scripst:1.1.0": *
```

to import the Scripst templates in your document.

This method does not require downloading the template files, just import them in the document.

---

After importing the template, create an **article** file in this way:

```
#show: scripst.with(
  title: [How to Use Scripst],
  info: [This is the article template],
  author: ("Author1", "Author2", "Author3"),
  time: datetime.today().display(),
  abstract: [Abstract],
  keywords: ("Keyword1", "Keyword2", "Keyword3"),
  contents: true,
  content_depth: 2,
  matheq_depth: 2,
  lang: "en",
)
```

See [Section 2](#) for the meaning of these parameters.

Then you can start writing your document.

## II Template Parameter Description

The Scripst template provides some parameters to customise the style of the document.

```
#let scripst(
  template: "article", // str: ("article", "book", "report")
  title: "",           // str, content, none
  info: "",            // str, content, none
  author: (),           // array
  time: "",            // str, content, none
  abstract: none,       // str, content, none
  keywords: (),         // array
  font_size: 11pt,     // length
  contents: false,     // bool
  content_depth: 2,    // int
  matheq_depth: 2,     // int: (1, 2)
  lang: "en",          // str: ("zh", "en", "fr", ...)
  body,
) = {
  ...
}
```

### 2.1 template

Parameter	Type	Optional Values	Default Value	Description
template	str	("article", "book", "report")	"article"	Template type

Currently, Scripst provides three templates: article, book, and report.

This template uses the article template.

- **article**: Suitable for daily documents, assignments, tiny notes, light papers, etc.
- **book**: Suitable for books, course notes, etc.
- **report**: Suitable for lab reports, papers, etc.

Passing other strings will cause a **panic**: "Unknown template!".

## 2.2 title

Parameter	Type	Default Value	Description
<code>title</code>	<code>content, str, none</code>	<code>""</code>	Document title

The title of the document. (If not empty) it will appear at the beginning and in the header of the document.

## 2.3 info


Parameter	Type	Default Value	Description
<code>info</code>	<code>content, str, none</code>	<code>""</code>	Document information

The information of the document. (If not empty) it will appear at the beginning and in the header of the document. It can be used as a subtitle or supplementary information for the article.

## 2.4 author

Parameter	Type	Default Value	Description
<code>author</code>	<code>array</code>	<code>()</code>	Document authors

The authors of the document. Pass a list of **str** or **content**.

 Note, if there is only one author, do not pass a **str** or **content**, but pass a list of one **str** or **content**, for example: **author**: ("Author",)

It will be displayed at the beginning of the article with `min(#authors, 3)` authors per line.

## 2.5 time

Parameter	Type	Default Value	Description
<code>time</code>	<code>content, str, none</code>	<code>""</code>	Document time

The time of the document. It will appear at the beginning and in the header of the document.

You can choose to use Typst's `datetime` to get or format the time, such as today's date:

```
datetime.today().display()
```

## 2.6 abstract

Parameter	Type	Default Value	Description
<code>abstract</code>	<code>content</code> , <code>str</code> , <code>none</code>	<code>none</code>	Document abstract

The abstract of the document. (If not empty) it will appear at the beginning of the document.

It is recommended to define a `content` before using the abstract, for example:

```
#let abstract = [
  This is a simple document template used to generate simple daily documents
  to meet the needs of documents, assignments, notes, papers, etc.
]

#show: scripst.with(
  ...
  abstract: abstract,
  ...
)
```

Then pass it to the `abstract` parameter.

## 2.7 keywords

Parameter	Type	Default Value	Description
<code>keywords</code>	<code>array</code>	<code>()</code>	Document keywords

The keywords of the document. Pass a list of `str` or `content`.

Like `author`, the parameter is a list, not a string.

Keywords will only appear at the beginning of the document if `abstract` is not empty.

## 2.8 font\_size

Parameter	Type	Default Value	Description
<code>font_size</code>	<code>length</code>	<code>11pt</code>	Document font size

The font size of the document. The default is **11pt**.

Refer to the **length** type values, you can pass **pt**, **mm**, **cm**, **in**, **em**, etc.

## 2.9 contents

Parameter	Type	Default Value	Description
<code>contents</code>	<b>bool</b>	<b>false</b>	Whether to generate a table of contents

Whether to generate a table of contents. The default is **false**.

## 2.10 content\_depth

Parameter	Type	Default Value	Description
<code>content_depth</code>	<b>int</b>	<b>2</b>	Depth of the table of contents

The depth of the table of contents. The default is **2**.

## 2.11 matheq\_depth

Parameter	Type	Optional Values	Default Value	Description
<code>matheq_depth</code>	<b>int</b>	<b>1, 2</b>	<b>2</b>	Depth of math equation numbering

The depth of math equation numbering. The default is **2**.

Generally, use **1** when there are no chapters, and use **2** when there are chapters.

## 2.12 lang

Parameter	Type	Default Value	Description
<code>lang</code>	<b>str</b>	<b>"zh"</b>	Document language

The document language. The default is **"zh"**.

Accepts [ISO\\_639-1](#) encoding format, such as **"zh"**, **"en"**, **"fr"**, etc.

## 2.13 body

When using `#show: scripst.with(...)`, the **body** parameter does not need to be passed manually. Typst will automatically pass the remaining document content to the **body** parameter.



## III Template Effect Display

### 3.1 Front Page

The beginning of the document will display the title, information, authors, time, abstract, keywords, etc., as shown at the beginning of this document.

### 3.2 Table of Contents

If the `contents` parameter is `true`, a table of contents will be generated, as shown in this document.

### 3.3 Fonts and Environments

Scriptst provides some commonly used fonts and environments, such as bold, italic, headings, images, tables, lists, quotes, links, math formulas, etc.

#### 3.3.1 Fonts

This is normal text. C'est un texte normal.

**This is bold text. C'est un texte en gras.**

*This is italic text. C'est un texte en italique.*

Install the CMU Serif font for better (LaTeX-like) display effects.

#### 3.3.2 Environments

##### 3.3.2.1 Headings

Level 1 headings are numbered according to the document language, including Chinese/Roman numerals/Greek letters/Kana/Numerals in Arabic/Hindi numerals, etc. Other levels use Arabic numerals.

##### 3.3.2.2 Images

The image environment will automatically number the images, as shown below:

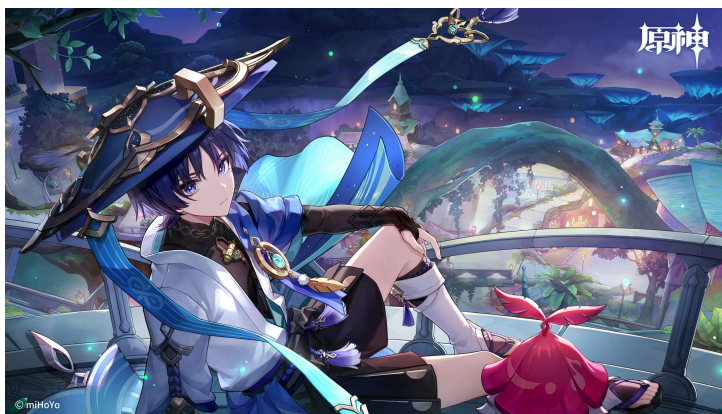


Figure 1: Little Scara

### 3.3.2.3 Tables

Thanks to the `tablem` package, you can write tables in Markdown style when using this template, as shown below:

```
#figure(
  three-line-table[
    | Name | Age | Gender |
    | --- | --- | --- |
    | Jane | 18 | Male |
    | Doe | 19 | Female |
  ],
  caption: [`three-line-table` table
example],
)
```

Name	Age	Gender
Jane	18	Male
Doe	19	Female

Table 1: `three-line-table` table example

```
#figure(
  tablem[
    | Name | Age | Gender |
    | --- | --- | --- |
    | Jane | 18 | Male |
    | Doe | 19 | Female |
  ],
  caption: [`tablem` table example],
)
```

Name	Age	Gender
Jane	18	Male
Doe	19	Female

Table 2: `tablem` table example

You can choose `numbering: none`, to make the table unnumbered, as shown above, the tables in the previous chapters did not enter the full text table counter.

### 3.3.2.4 Math Formulas

Math formulas have inline and block modes.

Inline formula:  $a^2 + b^2 = c^2$ .

Block formula:

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

$$\frac{1}{2} + \frac{1}{3} = \frac{5}{6} \quad (3.1)$$

are numbered.

Thanks to the `physica` package, Typst's math input method is greatly expanded while still retaining its simplicity:

$$\begin{aligned} \nabla \cdot \mathbf{E} &= \frac{\rho}{\varepsilon_0} \\ \nabla \cdot \mathbf{B} &= 0 \\ \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\ \nabla \times \mathbf{B} &= \mu_0 \left( \mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right) \end{aligned} \quad (3.2)$$

### 3.3.3 Lists

Typst provides a simple environment for lists, as shown:

```
- First item
- Second item
- Third item
```

- First item
- Second item
- Third item

```
+ First item
3. Second item
+ Third item
```

1. First item
3. Second item
4. Third item

```
/ First item: 1
/ Second item: 2
/ Third item: 23
```

**First item** 1  
**Second item** 2  
**Third item** 3

### 3.3.4 Quotes

```
#quote(attribution: "Einstein",
block: true)[
  God does not play dice with the
  universe.
]
```

*God does not play dice with the universe.*  
— *Einstein*

### 3.3.5 Links

```
#link("https://www.google.com/")
[Google]
```

Google

### 3.3.6 Hyperlinks and Citations

Use `<label>` and `@label` to achieve hyperlinks and citations.

## 3.4 #newpara() Function

By default, some modules do not automatically wrap. This is necessary, for example, if the explanation of the above math formula does not wrap.

But sometimes we need to wrap, and this is where the `#newpara()` function comes in.

Unlike the official `#parbreak()` function, the `#newpara()` function inserts a blank line between paragraphs, so it will start a new natural paragraph in any scenario.

Whenever you feel the need to wrap, you can use the `#newpara()` function.

## 3.5 countblock

`countblock` is a counter module provided by Scriptst to count certain countable content in the document.

The global variable `cb` records all available counters, and you can add a counter using the `add_countblock` function.

The default countblocks are

```
#let cb = (
  "thm": ("Theorem", color.blue),
  "def": ("Definition", color.green),
  "prob": ("Problem", color.purple),
  "prop": ("Proposition", color.purple-grey),
  "ex": ("Example", color.green-blue),
  "note": ("Note", color.grey),
  "cau": ("⚠", color.red),
)
```

These counters are already initialised, and you can use them directly.

**Note** Since Typst language functions do not have pointers or references, the passed variables cannot be modified. We can only modify variables through explicit return values and pass them to the next function. The author has not found a better method yet.

### 3.5.1 Creating and Registering countblock

At the same time, you can add (or override) a counter using the `add_countblock` function and then register this counter using the `register_countblock` function.

```
#let cb = add_countblock("test", "This is a test", teal)
#show: register_countblock.with("test")
```

After that, you can use the `countblock` function to count this counter.

### 3.5.2 Using countblock

Use the `countblock` function to create a block:

```
#countblock(
  name,
  subname,
  count: true,
  cb: cb,
)[ ... ]
```

where `name` is the name of the counter, `subname` is the name of the entry being created, `count` is whether to count, and `cb` is the list of counters. For example

```
#countblock("thm", subname: [_Fermat's Last Theorem_], cb)[

  No three $a, b, c$ in  $\mathbb{N}^{+}$  can satisfy the equation
  $
  a^n + b^n = c^n
  $
  for any integer value of $n$ greater than 2.
]
#proof[Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis
exiguitas non caperet.]
```

will create a theorem block and count it:

#### Theorem 3.1 *Fermat's Last Theorem*

No three  $a, b, c \in \mathbb{N}^{+}$  can satisfy the equation

$$a^n + b^n = c^n \tag{3.3}$$

for any integer value of  $n$  greater than 2.

*Proof.* Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.

■

where `subname` is required if passed.

You can also encapsulate it into another function:

```
#let test = countblock.with("test", cb)
```

For the `test` counter just created, you can use the `countblock` function to count:

```
#countblock("test", cb)[
  1 + 1 = 2
]

#test[
  1 + 2 = 3
]
```

**This is a test 3.1**     $1 + 1 = 2$

**This is a test 3.2**     $1 + 2 = 3$

The other default counters can also be used with the pre-packaged functions:

```
#definition(subname: [...])[]
#theorem(subname: [...])[]
#proposition(subname: [...])[]
#problem(subname: [...])[]
#note(count: false)[]
#caution(count: false)[]
```

### Definition 3.1

This is a definition, please understand it.

### Theorem 3.2

This is a theorem, please use it.

### Problem 3.1

This is a problem, please solve it.

### Proposition 3.1

This is a proposition, please prove it.

### Note

This is a note, please note it.



This is a caution, please be careful with it.

The numbering logic of these counters is:

- If there are no chapters, there is only one counter number
- If there are chapters, the counter number is **chapter number. the number of this type of block that has appeared in this chapter up to this block**

In this way, you can register and use any number of countblocks.

## IV Conclusion

The above documentation demonstrated Scripst, explained the template parameters, and showed the template effects.

I hope this document helps you better use Typst and Scripst.

You are also welcome to provide suggestions, improvements, and/or contribute code to Scripst.

Thank you for your support of Typst and Scripst!