いろいろできそうな Typst で遊ぼう

Typst の使い方

著者: ubanis

テスト版

目次

1 はじめに

1-1 Typst とは

Typst[1]は、科学技術文書の作成に特化した、現代的な組版(くみはん)システムです。Markdown のようにシンプルな構文と、LaTeX のようにパワフルで柔軟な表現力を兼ね備えています。リアルタイムでの高速なコンパイルが特徴で、効率的な執筆体験を提供します。

github ページ

https://github.com/typst/typst

ドキュメント

https://typst.app/docs/

1-2 Typst の利点

- **学習の容易さ:** 直感的で覚えやすい構文を採用しており、初心者でもすぐに使い始めることができます。
- **高速なコンパイル:** 変更を即座にプレビューに反映するため、トライ＆エラーが容易です。
- **強力なスクリプティング:** 変数、関数、ループ、条件分岐といったプログラミング機能が組み込まれており、動的で再利用性の高いコンテンツを作成できます。
- **統一されたエコシステム:** パッケージ管理機能が内蔵されており、外部のライブラリやテンプレートを簡単に導入できます。

1-3 インストール

Typst は、Web アプリケーションとしてブラウザ上で利用する方法と、ローカル環境にインストールして利用する方法があります。詳細は公式サイトを参照してください。

- **Web アプリケーション:** https://typst.app/
- **ローカルへのインストール:** https://github.com/typst/typst?tab=readme-ov-file#installation

以下は代表的なパッケージ管理ソフトでのインストール方法です。

1-3-1 winget(Windows)

```powershell
winget install --id Typst.Typst
```

1-3-2 scoop(Windows)

```powershell
scoop install typst
```

1-3-3 Cargo

```sh
cargo install --locked typst-cli
```

1-3-4 Snap(Linux)

```sh
sudo snap install typst
```

1-3-5 mise

```sh
mise use -g typst
```

1-4 はじめてのTypst

まずシンプルな以下のような内容のファイルを `example.typ` という名前で保存します。

```typst
= 見出しです

段落です。

- リスト1
- リスト2
- リスト3
```

以下のコマンドで対象のTypstファイルからpdfを出力します。

```sh
typst compile example.typ
```

出力されたpdfファイルを開いてみると以下のようになっているはずです。

見出しです

段落です。

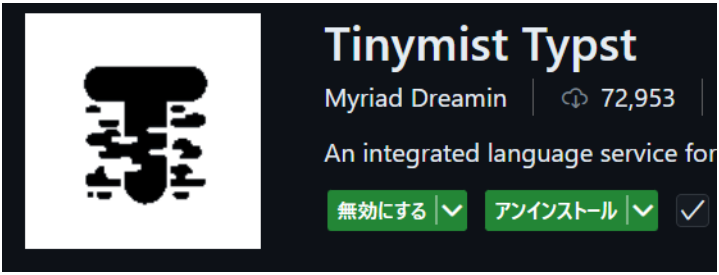
- リスト 1
- リスト 2
- リスト 3

以下のコマンドは対象の Typst ファイルを監視して変更があるたびにコンパイルをして pdf を更新し続けます。

```sh
typst watch example.typ
```

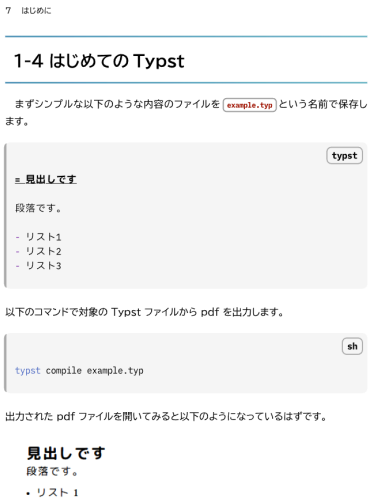
この状態で `example.typ` を編集するたびに pdf ファイルが更新されることになります。

1-5 vscode 拡張 Tinymist Typst の導入



vscode 上で Typst を快適に編集するための拡張機能 **Tinymist Typst** を導入すると、構文ハイライトや補完機能が利用可能になります。リアルタイムプレビュー機能もあり、Typst のドキュメントを編集しながら即座に結果を確認できます。

````
まずシンプルな以下のような内容のファイルを`example.typ`という名前で保存します。

```typst
= 見出しです

段落です。

- リスト1
- リスト2
- リスト3
```

以下のコマンドで対象の Typst ファイルから pdf を出力します。

```sh
typst compile example.typ
```

出力された pdf ファイルを開いてみると以下のようになっているはずです。

#image("../img/example.png", width: 5cm)

以下のコマンドは対象のTypstファイルを監視して変更があるたびにコンパイルをしてpdfを更新し続けます。

```sh
typst watch example.typ
```

この状態で`example.typ`を編集するたびに pdf ファイルが更新されることになります。
````



7　はじめに

1-4 はじめての Typst

まずシンプルな以下のような内容のファイルを example.typ という名前で保存します。

typst

= 見出しです

段落です。

- リスト1
- リスト2
- リスト3

以下のコマンドで対象の Typst ファイルから pdf を出力します。

sh

typst compile example.typ

出力された pdf ファイルを開いてみると以下のようになっているはずです。

見出しです

段落です。

- リスト 1

このようにとても編集しやすいのでぜひ導入しましょう。

1-5-1 プレビュー画面の表示

図 1: このアイコンをクリックする

vscode 上の右上にある虫眼鏡のついたプレビューアイコンをクリックすると現在のファイルのリアルタイムプレビュー画面が右側に表示されます。

プレビュー画面の文などの要素をクリックすると該当箇所のファイルの行へ移動で

き、ファイル内の要素をクリックするとプレビュー画面がその位置へスクロールします。

図 2: 外部ブラウザで開く

プレビュー画面の右上にある四角に右上矢印のアイコンをクリックするとプレビュー画面は外部ブラウザに表示されるようになります。

　クリックでジャンプする機能はブラウザ上でもそのまま利用できます。

2 基本的なマークアップ

2-1 見出し

`=` の数で見出しのレベルを表現します。

typst

```
= 章（レベル1）
== 節（レベル2）
=== 項（レベル3）
```

2-2 テキストの装飾

Typst では、簡単な記号でテキストを装飾できます。

強調（太字）

typst

> ***強調***

イタリック（斜体）

typst
```
_イタリック_
```

下線

typst
```
#underline[下線]
```

~~打ち消し線~~

typst
```
#strike[打ち消し線]
```

URL https://example.com

typst
```
URL https://example.com
```

\ 改行・エスケープ文字

```typst
このように改行できます。\ 文中では\ スペースを一つ空けます。\
\$←のようなTypstで利用されている文字を書きたい場合のエスケープ文字としても利用されます。
```

このように改行できます。
文中では
スペースを一つ空けます。
$←のような Typst で利用されている文字を書きたい場合のエスケープ文字としても利用されます。

`//` `/* */` コメント

```typst
//一行のコメント
/*
複数行の
コメント
*/
```

2-3 コードブロック

`（バッククォート 1 つ）で囲んだ文字はインラインコードになります。

`これはインライン`です。

```（バッククォート３つ）で囲んだ行はコードブロックを形成します。言語を指定するとシンタックスハイライトが適用されます。

```c
//これはブロックです。
//シンタックスハイライトが適用されています。
#include <stdio.h>

void main(int argc, char *argv[])
{
  char *str = "hello world!";
  puts(str);
}
```

2-4 リスト

`-`（ハイフン）や `+`（プラス）でリストを作成します。インデントすることでネストも可能です。

```typst
- 順序なしリスト
  - ネストされたリスト
- 順序付きリスト
  + 番号は自動で振られます
  + 2番目の項目
```
```

- 順序なしリスト
  - ネストされたリスト
- 順序付きリスト
  1. 番号は自動で振られます
  2. 2 番目の項目

2-5 用語リスト

typst

/ **Typst** : 今注目されている新しい組版システム。2019年に開発が開始され、2023年3月にベータ版が一般公開された。Rustで書かれており、シンプルな文法やコンパイルの速さが利点。

/ **Rust** : メモリ安全性を保証しつつ、低レベルのシステムプログラミングが可能な言語。所有権システムを採用しており、ガベージコレクタ#footnote("コンピュータプログラムが動的に確保したメモリ領域のうち、不要になった領域を自動的に解放する機能です。")がなくてもメモリリークやデータ競合を防ぐ。

/ **ねこ** : @nyan-image

Typst

今注目されている新しい組版システム。2019 年に開発が開始され、2023 年 3 月にベータ版が一般公開された。Rust で書かれており、シンプルな文法やコンパイルの速さが利点。

Rust

メモリ安全性を保証しつつ、低レベルのシステムプログラミングが可能な言語。所有権システムを採用しており、ガベージコレクタ[†1] がなくてもメモリリークやデータ競合を防ぐ。

ねこ

図 1

2-6 数式

`$` で数式を囲みます。1 行でインライン数式、複数行でブロック数式になります。

typst

```
/ インライン数式 : $E = m c^2$ はインライン数式です。

/ ブロック数式 :
  これはブロック数式です。
  $
```

†1　コンピュータプログラムが動的に確保したメモリ領域のうち、不要になった領域を自動的に解放する機能です。

```
sum_(i=1)^n i = (n(n+1)) / 2
$
```

インライン数式

$E = mc^2$ はインライン数式です。

ブロック数式

これはブロック数式です。

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

2-7 参照

`<ラベル名>` で要素にラベルを付け、`@ラベル名` で参照します。数式や図、セクションなど、あらゆる要素にラベルを付けられます。

typst

```
#figure(
  image("../img/nyan4.jpg", width: 3cm),
  caption: [参照されるねこ],
) <nyan-image>
#align(center)[@nyan-image をこのように参照できます。]
```

図 1: 参照されるねこ

図 1 をこのように参照できます。

3 関数

3-1 Typst における関数

`#` で始まる文は関数を表します。内蔵の関数や以下のように `#let` を使って自作の関数を定義し利用することができます。

typst

```
#let test(body) = { [渡された文字列は#body] }
- #test("テスト")
- #test[#text(fill: red, weight: "bold", [test])]
```

- 渡された文字列はテスト
- 渡された文字列は **test**

引数の渡し方は複数あります。以下は body に引数が割り当てられます。

typst

```
#test("テスト")
```

このようにマークアップ文全体をを渡すこともできます。これも自動的に body に割り当てられます。

typst

```
#test[#text(fill: red, weight: "bold", [test])]
```

以下のように複数の引数がある場合も自動的に body に割り当てられます。引数の初期値も設定でき省略時にこれが参照されます。

typst

```
#let test2( var: false, body) = {
  if var {
    [varはTrueです。]
  } else {
    [varはFalseです。]
  }
  body
}

- #test2[引数]
- #test2(var: true)[引数]
```

- var は False です。引数
- var は True です。引数

{ } で区切られたブロック内はコードブロックなので関数に # をつけずに呼び出します。マークアップ文を使う場合は [ ] で囲みます。

コンテントブロック(Content block) `[...]`

関数は `#` をつけて使う。`#text(fill: red)[*Hey* there!]`

コードブロック(Code block) `{...}`

関数は `#` をつけずに使う。`text(fill: red)[*Hey* there!]`

3-2 マークアップ関数の一覧

目次 `#outline(target: heading, depth: 2)`

target で作りたい目次の要素を指定する。`depth` は目次を作るレベルの深さ。

水平方向のスペース `#h(1fr)`

—1fr—　　—1fr—　　—1fr—

垂直方向のスペース `#v(0.2em)`

——

——

中央揃え `#align(center)[テキスト]`

テキスト

取り消し線 `#strike[文字列]`

文字列

下線 `#underline[文字列]`

<u>文字列</u>

大文字化 `#upper[apple]`

APPLE

小文字化 `#lower[APPLE]`

apple

スモールキャピタル `#smallcaps[Smallcaps]`

Smallcaps

上付き文字 `2#super[2]`

2^{2}

下付き文字 `2#sub[2]`

2_{2}

ハイライト `#highlight[文字列]`

文字列

引用 `#quote(block: true, attribution: [名前])[引用文字列]`

> 引用文字列
>
> — 名前

囲み `#box(stroke: 0.5pt + black,outset: 0.3em)[文字列]`

文字列

ダミーテキストを 10 単語生成 `#lorem(10)`

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

今日の日付を取得 `#datetime.today()`

`datetime(year: 2025, month: 7, day: 16)`

水平線 `#line(stroke: 1pt + black, start: (5%,0%), end: (95%,0%))`

改ページ `#pagebreak()`

この関数から下は次のページになります。

3-3 スクリプト

Typst はスクリプト言語としての機能も持っています。変数、ループ、条件分岐などを利用して、コンテンツを動的に生成できます。

3-3-1 for 関数

typst

```
#let users = ("Alice", "Bob", "Charlie")
#for user in users {
```

```
  [こんにちは、#text(fill: red)[#user] さん！]
}
```

こんにちは、Alice さん！こんにちは、Bob さん！こんにちは、Charlie さん！

3-3-2 if 関数

typst

```
#let a=32
#let b=64
#align(center)[#if a < b { [aはbより小さいです。] }]
```

a は b より小さいです。

3-3-3 単純な画像表示関数を定義

typst

```
#let nyan(size: 5cm) = {image("../img/nyan4.jpg", width:size)}
#align(center)[#nyan(size: 2.5cm)]
```

3-4 image 関数

`#image()` 関数で画像を挿入します。`#figure()` と組み合わせることで、キャプションや図番号を付与できます。

typst

```
#figure(
  image("../img/nyan4.jpg", width: 3cm),
  caption: [ねこです]
)
```

図 1: ねこです

3-5 図形関数

`#rect()` や `#circle()` などの関数で、基本的な図形を簡単に描画できます。

typst

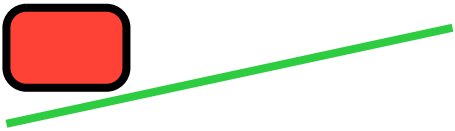
```
#align(center)[
#grid(columns: 3,
  [#square(size: 1cm, fill: blue)],
  [#circle(radius: 0.5cm, fill: red)],
  [#polygon(
    (0cm, 1cm), (0.5cm, 0cm), (1cm, 1cm),
    fill: green,
  )]
)]
```

3-6 色の指定方法

関数の引数に `color` がある場合、色を指定できます。`fill` は塗りつぶしです。`stroke` は枠線です。`3pt + black` のように `+` で属性を追加できます。

typst

```
#rect(fill: red, stroke: 3pt + black, radius: 1em)
#line(stroke: 3pt + green, start: (0%,0%), end: (50%, -4em))
```



`rgb("#000000")`のように RGB 値で指定することもできます。透明度を含めた`rgb("#000000aa")`のような指定方法もあります。

typst

```
#grid(columns: 2,
polygon(
    (0cm, 1cm), (0.5cm, 0cm), (1cm, 1cm),
    fill: rgb("#0000cc"),
  ),
circle(radius: 0.5cm , fill: rgb("#0000cc55"))
)
```

グラデーションを指定するには`gradient`を利用します。グラデーションの種類を`gradient.linear`のように指定します。グラデーションの方向は`angle: 90deg`のように角度で指定します。

typst

```
#rect(fill: gradient.linear(red, blue,angle: 90deg), radius: 1em,)
```

3-7 table 関数

`#table()` 関数で柔軟な表を作成できます。

typst

```
#table(
  columns: (auto, 1fr, 1fr),
  stroke: 0.5pt,
  align: (left),
  [*No.*],[*項目*],[*説明*],
  [1], [Typst], [組版システム],
  [2], [Markdown], [軽量マークアップ言語],
  [3], [LaTeX], [組版システム],
)
```

No.	項目	説明
1	Typst	組版システム
2	Markdown	軽量マークアップ言語
3	LaTeX	組版システム

3-8 columns 関数

`#columns()` 関数で多段組を実現できます。`#colbreak()` で段を区切ります。

```typst
#import "@preview/roremu:0.1.0": roremu
#columns(2)[
  #roremu(100)
  #colbreak()
  #roremu(100)
]
```

吾輩は猫である。名前はまだ無い。どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれ

吾輩は猫である。名前はまだ無い。どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれ

このように長い文章は折り返され、指定した段数で区切られます。`#lorem()`（ここで使っているのは外部パッケージの`#roremu`）は指定された文字数の例文を表示する関数です。

3-9 grid 関数

```typst
#set rect(
    inset: 8pt,
    fill: rgb("e4e5ea"),
```

```
    width: 100%,
  )

#grid(
    columns: (60pt, 1fr, 2fr),
    rows: (auto, 60pt),
    gutter: 3pt,
    rect[Fixed width, auto height],
    rect[1/3 of the remains],
    rect[2/3 of the remains],
    rect(height: 100%)[Fixed height],
    grid.cell(
      colspan: 2,
      image("../img/nyan4.jpg", width: 100%),
    ),
  )
```

Fixed width, auto height	1/3 of the remains	2/3 of the remains
Fixed height		

3-10 footnote 関数

脚注をつけることができます。

typst

```
Astro#footnote("SSGだけでなくSSRもできる静的サイトジェネレーター")
もありますがTypstは使えません。
```

Astro[†1] もありますが Typst は使えません。

3-11 要素の移動や回転

ある要素を移動させたい場合以下のような関数が利用できます。他の要素の位置に注意して使用してください。

3-11-1 特定位置への要素の表示

`#place` 関数を使用することでページ上の特定位置に要素を表示することができます。要素の中ならばその要素の相対位置になります。

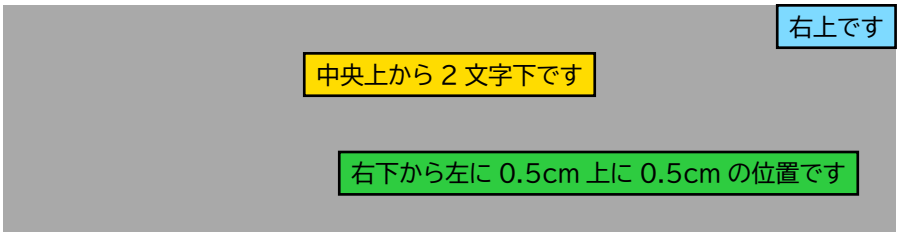
`#block` 関数でブロック領域を作りその中で `#place` 関数を使ってみます。灰色の部分がブロック領域です。

typst

```
```

†1　SSG だけでなく SSR もできる静的サイトジェネレーター

```
#let mybox(body, color: white)={
  box(stroke: 1pt, fill: color, inset: 0.5em, body)
}
\
#block(height: 3cm, width: 100%, fill: gray)[
#place(top+right)[#mybox([右上です], color: aqua)]
#place(top+center, dy: 2em)[#mybox([中央上から2文字下です], color:
yellow)]
#place(bottom+right, dx: -0.5cm, dy: -0.5cm)[#mybox([右下から左に
0.5cm上に0.5cmの位置です], color: green)]
]
```



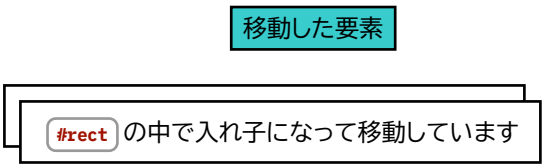
3-11-2 現在位置から指定位置への要素の移動

`#move` 関数を使用すると要素の現在位置から指定された位置に要素を移動できます。

typst

```
#let mybox(body, color: white)={
  box(stroke: 1pt, fill: color, inset: 0.5em, body)
}
#move(dx: 3cm)[#mybox([移動した要素], color: teal)]
```

```
#rect(inset: 0pt, move(
  dx: 6pt, dy: 6pt,
  rect(
    inset: 8pt,
    fill: white,
    stroke: black,
    [`#rect`の中で入れ子になって移動しています]
  )
))
```



3-11-3 要素を回転させる

#rotate 関数を利用して要素を指定した角度に回転させることができます。

typst

```
#set text(fill: white)
#align(center,block(fill: black, width: 100%,
  [
    #place(left + horizon, dx: 6em)[
      #rotate(-70deg)[*ローテート!*]]
    #rotate(-10deg)[
      #rotate(45deg)[#block(radius: 100%, clip: true)[
        #image("../img/nyan4.jpg", width: 3cm)
        ]
      ]
      このように要素が
```

```
      #rotate(30deg)[すべて]
      指定された角度`deg`に回転します。
      #rotate(25deg)[
        #move(dx: 7em, dy: -8em)[入れ子になっていても大丈夫です。]
        ]
    ]
  ]
))
```

3-12 ファイルの分割

`#for` 関数と `#include` 関数を使って分割されたファイルを読み込みします。

複数のファイルに分割することで項目ごとに編集管理しやすくできます。

```
                                                    typst
```

```
// ファイルのパスを配列にする
#let chapters = (
  "content/text-format.typ",
  "content/heading.typ",
  "content/list.typ",
  "content/codeblock.typ",
  "content/formula.typ",
)

= ここからchaptersが配列順にそれぞれ並ぶ

#for chapter in (chapters) {
  include(chapter)
}
```

3-13 リストのループ

別ファイルからリストを読み込み `#for` ループを利用して用語リストとテーブルに表示してみます。`#table` 関数では `..#for` と書くことで配列を渡します。

3-13-1 import される Typst ファイル

typst

```
#let locales=(
  ( name:"日本",str:"首都は東京です。",),
  ( name:"アメリカ",str:"首都はワシントンD.C.です。",),
  ( name:"フランス",str:"首都はパリです。",),
  ( name:"イギリス",str:"首都はロンドンです。",),
```

```
  ( name:"ドイツ",str:"首都はベルリンです。",),
  ( name:"中国",str:"首都は北京です。",),
  ( name:"韓国",str:"首都はソウルです。",),
)
```

3-13-2 コード

typst

```
#import "../import/locales.typ": locales

#for c in locales {
  terms.item(text(c.name),text(c.str))
}

#table(
  columns: (auto, 1fr),
  stroke: 1pt+gray,
  align: (left),
  [*国名*],[*首都*],
  ..for (name,str) in locales {
    (name,str)
  }
)
```

3-13-3 実行結果

日本

首都は東京です。

アメリカ

首都はワシントン D.C.です。

フランス

首都はパリです。

イギリス

首都はロンドンです。

ドイツ

首都はベルリンです。

中国

首都は北京です。

韓国

首都はソウルです。

国名	首都
日本	首都は東京です。
アメリカ	首都はワシントン D.C.です。
フランス	首都はパリです。
イギリス	首都はロンドンです。
ドイツ	首都はベルリンです。
中国	首都は北京です。

韓国	首都はソウルです。

3-14 外部データの読み込み

Typst は以下のデータを読み込むことができる。

- cbor
- csv
- json
- read
- toml
- xml
- yaml

3-14-1 読み込むデータ

json

```
[
  {
    "name": "ちんぽねこ",
    "path": "../img/e670ceab9f15d95ce92d718ba046aaac3142aac3.jpg",
    "description": "クリスマスにゃんこだにゃん"
  },
```

```
  {
    "name": "きつね",
    "path": "../img/2022-0106-01.jpg",
    "description": "すけべなところを見せつけてくるきつね"
  },
  {
    "name": "ティファ",
    "path": "../img/c2b7d960c6205cb3d973fe0b3c51b477ecf75202.jpg",
    "description": "FFでもトップのすけべ担当"
  },
  {
    "name": "ブリジット",
    "path": "../img/1e4697102be08c9174a60f109b4e6558e838bf81.jpg",
    "description": "格ゲー一番のおすちんぽ"
  }
]
```

3-14-2 コード

typst

```
#let columnimage(data) = columns(
  2,
  for img in data{
    block(
      width: 100%,
      clip: true,
      square(
        radius: 1em,
        width: 100%,
        height: auto,
        inset: 0.5cm,
        fill: if img.name == "ちんぽねこ" {
```

```
          yellow
        } else {
          aqua
        },
        highlight(fill:rgb("#77ffff"),
text(strong(underline(img.name))))
        +"\n"+
        text(img.description)
        +align(center, rotate(8deg, image(img.path, fit:"cover",
width: 80%))),
      )
    )
  }
)

#columnimage(json("../json/animals.json"))
```

3-14-3 実行結果

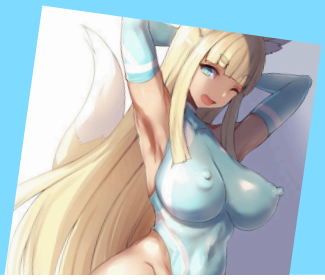
ちんぽねこ

クリスマスにゃんこだにゃん

きつね

すけべなところを見せつけてくる

きつね



ティファ

FF でもトップのすけべ担当

ブリジット

格ゲー一番のおすちんぽ

4 スタイリング

4-1 `#set` を利用したスタイリング

`#set` を利用して要素のスタイルを設定できます。

`par` は段落を設定します。`text` はテキストを設定します。

typst

```
#import "@preview/roremu:0.1.0": roremu
#set par(spacing: 1em, leading: 1em)
#set text(weight: "bold", size: 0.8em)
#columns(2)[
#roremu(400)
]
```

吾輩は猫である。名前はまだ無い。どこで生れたかとんと見当がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番獰悪な種族であったそうだ。この書生というのは時々我々を捕えて煮て食うという話である。しかしその当時は何という考もなかったから別段恐しいとも思わなかった。ただ彼の掌に載せられてスーと持ち上げられた時何だかフワフワした感じがあったばかりである。掌の上で少し落ちついて書生の顔を見たのがいわゆる人間というものの見始であろう。この時妙なものだと思った感じが今でも残っている。第一毛をもって装飾されべきはずの顔がつるつるしてまるで薬缶だ。その後猫にもだいぶ逢ったがこんな片輪には一度も出会わした事がない。のみならず顔の真中があまりに突起している。そうしてその穴の中から時々ぷうぷうと煙を

`#set` の適用範囲はファイルの終端かブロックの終わりまでです。

typst

```
#[
#set list(marker: [★], indent: 2em)
- ここは適用される
]
- こちらは適用されない
```

★ ここは適用される

- こちらは適用されない

4-1-1 ページのスタイリング

`#set page()` ルールで、用紙サイズ、余白、ヘッダー、フッターなどを設定できます。

typst

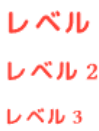
```
#set page(
  paper: "a4",
  margin: (x: 2.5cm, y: 3cm),
  header: align(right)[Typst Manual],
  footer: align(center)[#counter(page).display("1")]
)
```

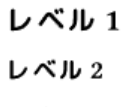
4-2 `#show` を利用したスタイリング

`#show` を使用すると、要素のスタイルを細かくカスタマイズできます。`#[...]` は無名のコードブロックです。

typst

```
#[
  #show heading: set text(fill: red)
  = レベル1
  == レベル2
  === レベル3
]
ブロックの外なので通常です
= レベル1
== レベル2
===  レベル3
```





コロン `:` の後に関数をつけてさらに細かくカスタマイズすることもできます。`it` は要素の実体を表します。

typst

```
#[
  #show heading: it => [
    #set text(fill: green)
    #align(center)[#it]
  ]
= レベル1
== レベル2
=== レベル3
]
```

要素の中の特定要素を `where` で指定することもできます。

typst

```
#[
  #show heading.where(level: 2): it => [
    #set text(fill: fuchsia)
    #align(center)[#it]
  ]
  = レベル1
  == レベル2
  === レベル3
]
```

単純な文字列の置き換えは以下の方法が利用できます。

typst

```
#block(stroke: 1pt + gray, inset: 5mm)[
  #show "日本語": "nihongo"
  日本語という文字列がこのように変化しています。
]
```

nihongo という文字列がこのように変化しています。

#show に regex を利用して特定文字列を対象にすることもできます。

typst

```
#block(stroke: 1pt + gray, inset: 5mm)[
  #show regex("apple"): [ジョブズ]
  apple computer
]
```

ジョブズ computer

typst

```
#block(stroke: 1pt + gray, inset: 5mm)[
  #show regex("ジョブズ"): it => [
    #rotate(30deg)[#strong(it)]
  ]
```

```
  apple ジョブズ
]
```

4-3 `#show:` と `with` を利用して外部のテンプレートを利用する

`#set` や `#show` で設定したスタイリングはそのファイル内かブロック内で有効になりますが､これを外部ファイルに記述してそれを `#import` して利用することができます。

`layout.typ` というファイルを作成し、`#let` を使って名前はなんでもよいのですが `layout` を定義します。

そして `=` に続くコードブロック内でスタイリングをします。

引数は自由に定義して問題ありません｡コロン `:` の後に続く要素はデフォルト値になります。

typst

```
#let layout(
  body,
  title: "Typstの使い方",
  author: "ubanis",
  pagesize: "a5",
```

```
  column: 1,
  fontname: "Noto Sans CJK JP",
  fontsize: 8pt,
  language: "ja",
) = {
  // ドキュメントの設定
  set document(
    title: _title,
    author: author,
  )
  // ページの設定
  set page(
    fill: none,
    paper: pagesize,
    flipped: false,
    binding: left,
    margin: (
      top: 25mm,
      bottom: 15mm,
      inside: 15mm,
      outside: 15mm,
    ),
    columns: column,
    numbering: "1",
    )
  // フォントの設定
  set text(
    font: fontname,
    lang: language,
    size: fontsize,
  )
  // 本文
  {//このコードブロックはなくてもよい
    body
  }
```

```
}// layoutの終わり
```

必ず関数の最後の方に `body` を入れてください。

そしてこれを本体の文書から `#import` を使って読み込みます。

`#import "layout.typ"` の後に続く `:` はそのファイルから読み込む関数を指定します。`*` は全てを読み込むことを表しています。

特定の関数を読み込むならば `#import "layout.typ": layout` のように指定します。

そしてその下で `#show` に `:` を付けて `layout.with` 関数を呼び出します。

typst

```
#import "layout.typ": *
#show: layout.with(
  title: "いろいろできそうなTypstで遊ぼう",
  author: "ubanis",
  pagesize: "a5",
  column: 1,
  fontsize: 9pt,
  fontname: "BIZ UDPGothic",
)

= ここから本文です

`layout`で設定した要素がここに適用されます。
この部分全てがlayoutの引数`body`に入っています

- リスト1
- リスト2
- リスト3
```

コードにあるように `#show: layout.with(...)` 以降のファイル内は `layout.typ` の `#layout` 内で設定したスタイルが適用されます。

他に `#include` 関数もあり、別ファイルの内容がそのまま同じファイル内に取り込まれるため以下のようにした場合も別ファイルにスタイルが適用されます。

typst

```
#show: layout.with(..) // 省略

= ここから本文です

`layout`で設定した要素がここに適用されます。
この部分全てがlayoutの引数`body`に入っています

- リスト1
- リスト2
- リスト3

#include "next.typ" //このファイルの中身もlayoutのスタイルが適用される。
```

`layout.typ` の `body` の前にコンテンツを書けばそれはこの本文の前に置かれます。`#outline` を使って目次を載せたり引数を活用して表紙を作成したりするのもよいでしょう。

typst

```
// ここまで省略
// タイトルページ
{
  v(3cm)
  set text(size: 1.5em)
  align(center, title)
  align(bottom+center, author)
```

```
    pagebreak()//改ページ
  }
  // 目次
  {
    set text(size:0.9em)
    outline(
      title: [目次],
      depth: 2,
      indent: 2em,
    )
    pagebreak()//改ページ
  }
  // 本文
  {//このコードブロックはなくてもよい
    body
  }
}// layoutの終わり
```

5 外部パッケージ

5-1 codly コードブロック装飾

https://typst.app/universe/package/codly

以下のようにインポートと初期化を行うことでコードブロックのスタイルが変化します。

typst
```
#import "@preview/codly:1.3.0": *
#import "@preview/codly-languages:0.1.1": *
#show: codly-init.with()
```

Rust の例

rust
```
pub fn main() {
    println!("Hello, world!");
}
```

一部分のハイライト

typst
```
#codly(highlights: (
  (line: 4, start: 2, end: none, fill: red),
  (line: 5, start: 13, end: 19, fill: green, tag: "(a)"),
  (line: 5, start: 26, fill: blue, tag: "(b)"),
```

```
5 ))
```

```py
1 def fib(n):
2   if n <= 1:
3     return n
4   else:
5     return fib(n - (a)1) + fib(n - 2)(b)
6 print(fib(25))
```

5-2 Rubby ルビをふる

https://typst.app/universe/package/rubby/

5-2-1 初期化

```typst
#import "@preview/rubby:0.10.2": get-ruby

#let ruby = get-ruby(
  size: 0.5em,          // Ruby font size
  dy: 0em,              // Vertical offset of the ruby
  pos: top,             // Ruby position (top or bottom)
  alignment: "center", // Ruby alignment ("center", "start",
"between", "around")
  delimiter: "|",       // The delimiter between words
  auto-spacing: true,  // Automatically add necessary space around
words
)
```

5-2-2 ルビのふりかた

```typst
ルビはRubbyを#ruby[り|よう][利|用]する。
```

ルビは Rubby を利用(りよう)する。

5-3 wrap-it 回り込み

https://typst.app/universe/package/wrap-it/

5-3-1 初期化

```typst
#import "@preview/wrap-it:0.1.1": wrap-content
```

5-3-2 使用例

```typst
#let wrap_fig = [
  #figure(
  image("../../img/nyan4.jpg",width:5cm),
  caption: [画像],
)<fig:nyan4>
```

```
]

#let wrap_body = [#lorem(100)]
#wrap-content(wrap_fig, wrap_body,align: bottom + left, column-
gutter: 1em)
* @fig:nyan4 *
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

図 1: ラベル付き画像

図 1

5-4 showybox 様々なボックス描画

https://typst.app/universe/package/showybox

様々な枠線を作ることができるパッケージです。

typst

```
#import "@preview/showybox:2.0.4": showybox
```

typst

```
#showybox(
  [Hello world!]
)
```

Hello world!

typst

```
#showybox(
  frame: (
    border-color: red.darken(50%),
    title-color: red.lighten(60%),
    body-color: red.lighten(80%)
  ),
  title-style: (
    color: black,
    weight: "regular",
    align: center
  ),
  shadow: (
    offset: 3pt,
  ),
  title: "Red-ish showybox with separated sections!",
  lorem(20),
  lorem(12)
)
```

Red-ish showybox with separated sections!

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.

typst

```
#showybox(
  frame: (
    dash: "dashed",
    border-color: red.darken(40%)
  ),
  body-style: (
    align: center
  ),
  sep: (
    dash: "dashed"
  ),
  shadow: (
    offset: (x: 2pt, y: 3pt),
    color: yellow.lighten(70%)
  ),
  [This is an important message!],
  [Be careful outside. There are dangerous bananas!]
)
```

This is an important message!

Be careful outside. There are dangerous bananas!

6 付録

6-1 地震情報を掲載する

気象庁の地震情報を取得できるアドレス https://www.jma.go.jp/bosai/quake/data/list.json から json ファイルを取得してテーブル表示します。

6-1-1 json ファイルの取得

```sh
wget https://www.jma.go.jp/bosai/quake/data/list.json
```

リストは 1000 件と長すぎるため表示する範囲は `slice` で限定しました。

```typst
#let earthquake-list(num) = {
  set text(
    size: 0.8em,
  )
  let earthquakes = json("json/list.json")
  [== 最近の地震 #num 件]
  table(
```

```
    stroke: 0.5pt,
    align: left,
    columns: (auto,auto,auto,auto),
    [*地震発生時刻*], [*震源地*], [*マグニチュード*], [*最大震度
*],
    ..for (at,anm,mag,maxi) in earthquakes.slice(0,num){
      (at,anm,mag,maxi)
    }
  )
}

#earthquake-list(10)
```

6-1-1-1 最近の地震 10 件

地震発生時刻	震源地	マグニチュード	最大震度
2025-07-16T04:48:00+09:00	トカラ列島近海	2.4	1
2025-07-16T04:20:00+09:00	トカラ列島近海	3.3	2
2025-07-16T02:43:00+09:00	トカラ列島近海	3.5	2
2025-07-16T02:18:00+09:00	トカラ列島近海	2.5	1
2025-07-16T02:15:00+09:00	トカラ列島近海	4.6	4
2025-07-16T02:15:00+09:00	トカラ列島近海	4.6	
2025-07-16T02:15:00+09:00			4
2025-07-16T02:08:00+09:00	トカラ列島近海	2.4	1
2025-07-16T00:49:00+09:00	広島県南東部	3.5	2
2025-07-16T00:10:00+09:00	トカラ列島近海	2.5	1

参照先

[1]「Typst Overview」. [Online]. 入手先: https://typst.app/docs/