

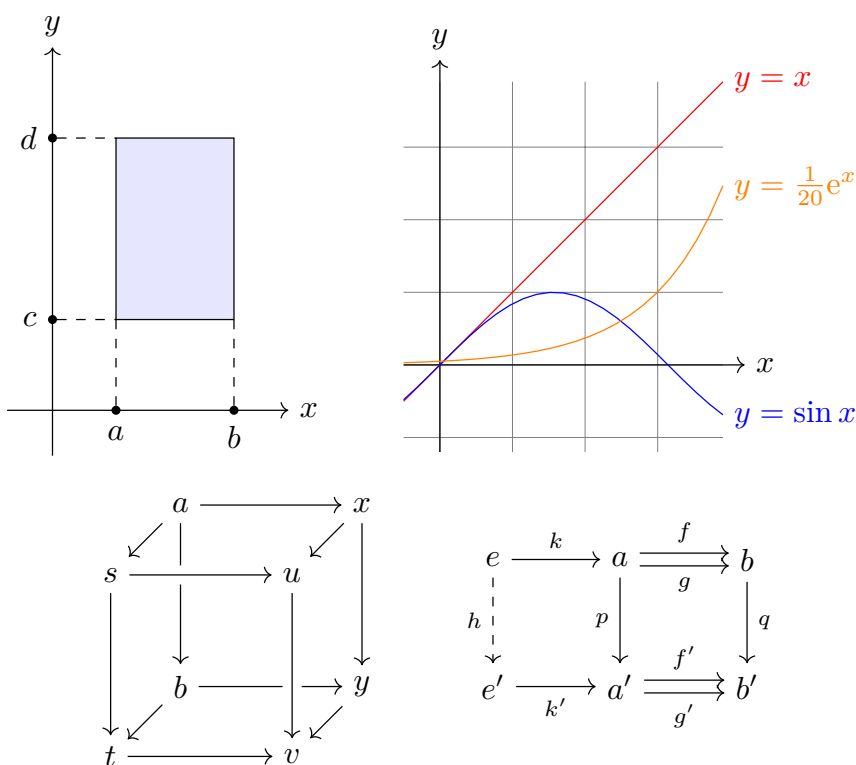
TikZ の使い方

alg-d

<https://alg-d.com/math/>

2025 年 7 月 5 日

TikZ とは $\text{T}_\text{E}\text{X}$ のパッケージの 1 つであり, これを使うと関数のグラフや可換図式など様々な絵を $\text{T}_\text{E}\text{X}$ 上で描くことができる. 例えば次のような絵を描くことができる.



TikZ には公式のマニュアル [1] があり, Windows で $\text{T}_\text{E}\text{X}$ Live を使っている場合はコマンドプロンプトで「`texdoc tikz`」を実行すれば読むことができる. しかしこの公式マニュアルは量が膨大で (1321 ページある) 初めはどこから手を付けたらいいか分かりづらい. そこで必要な部分を抜き出して解説したのがこの PDF である. 内容としては第 1 章が基本的な知識であり, とりあえずこの辺りを知っておけば, 後は自分で調べながら

TikZ を使うことが出来ると思う．第 2 章はより高度な内容ではあるが，知っておくと良さそうなものを記載した．なお，どこから抜き出したかすぐ分かるよう，各所に参照元を ([1] 節番号) の形式で書いてある．

目次

第 1 章	基本的な知識	6
1.1	TikZ	6
1.1.1	tikzpicture 環境	6
1.1.2	略記法	8
1.1.3	オプション引数について	8
1.2	座標系	9
1.3	基本的なオペレーション	10
1.3.1	Move-To オペレーション	10
1.3.2	Line-To オペレーション	10
1.3.3	rectangle オペレーション	11
1.3.4	circle オペレーション	12
1.3.5	grid オペレーション	13
1.3.6	cycle	14
1.4	パスのオプション	15
1.5	node オペレーション	28
1.5.1	ノードの形	29
1.5.2	at オプション	30
1.5.3	pos オプション	30
1.5.4	ノードに名前を付ける	31
1.5.5	[shape=coordinate]	34
1.5.6	ノードのオプション	35
1.5.7	ノードに複数行の文字を書く	39
1.6	Arrow Tips	40
1.7	baseline オプション	41

1.8	オプション指定時に使える便利な機能	42
1.8.1	スタイル	43
1.8.2	scope 環境	44
1.8.3	every	48
1.9	圏論で図式を描く例	50
第 2 章	より高度な内容	52
2.1	newcommand について	52
2.2	色の設定について	53
2.2.1	色の指定の仕方について	53
2.2.2	色の定義と形式について	55
2.3	bounding box	57
2.4	座標について	58
2.4.1	極座標系について	58
2.4.2	相対的な座標指定	59
2.4.3	座標へのオプション	60
2.5	その他のオペレーション	61
2.5.1	Curve-To オペレーション	61
2.5.2	to オペレーション	62
2.5.3	edge オペレーション	65
2.5.4	arc オペレーション	65
2.5.5	parabola オペレーション	66
2.5.6	sine オペレーション	67
2.5.7	decorate オペレーション	67
2.5.8	pic オペレーション	70
2.5.9	plot オペレーション	71
2.5.10	child オペレーション	72
2.5.11	foreach オペレーション	72
2.5.12	let オペレーション	72
2.6	label オプションと pin オプション	73
2.6.1	label オプション	73
2.6.2	pin オプション	75
2.6.3	The Quotes Syntax	76

2.7	perpendicular 座標系	76
2.8	座標計算	77
2.9	tangent 座標系	80
2.10	パスの交点を参照する	82
2.11	計算式について	83
2.12	レイヤーについて	91
2.13	pattern ライブラリ	92
2.14	graphs ライブラリ	93
2.15	circuits ライブラリ	93
2.16	through ライブラリ	95
2.17	beamer について	95
2.18	tikz-cd について	98
2.19	Magnifying Parts of Pictures	99
2.20	tikzmath コマンド	99
2.21	remember picture オプション	100
参考文献		101
索引		102

第 1 章

基本的な知識

1.1 TikZ ([1]1)

TikZ は TikZ ist *kein* Zeichenprogramm の略である (英語にすると TikZ is not a drawing program). TikZ を使用するためには, 他のパッケージを使うときと同じように `\usepackage{tikz}` と書く^{*1}. 基本的にはこれだけで TikZ は動作するが, 時々何も表示されない人がある. これは大抵, グローバルオプション (`\documentclass` のオプション) に `dvipdfmx` が設定されていないことが原因なので何も表示されない人は試してみてほしい (つまり例えば `\documentclass[dvipdfmx]{jsarticle}` のように書く)^{*2}.

1.1.1 tikzpicture 環境 ([1]12.2.1, 14)

TikZ では, 基本的に `tikzpicture` 環境を使用し, この環境の中に `\path ~~~~;` という形式で記述することで図形を描いていく (セミコロンが必要なことに注意). `\path` は複数書くことが可能である. よって実際に使用する際には次のような形式になる.

```
\begin{tikzpicture}
\path ~~~~;
\path ~~~~;
\path ~~~~;
\end{tikzpicture}
```

~~~~ の部分には「オペレーション」と呼ばれるコマンドを書く. 詳細な説明は以降の節ですが, ここでは例として `\path (0,0) --(2,1);` を挙げておく. これは (0,0)

---

<sup>\*1</sup> 圏論の図式など, 矢印を描く人は一緒に `\usetikzlibrary{arrows.meta}` も書くことを推奨する. 詳しくは [1]16.1 や [3] を参照.

<sup>\*2</sup> 詳しくは [4] を参照.

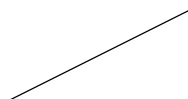
と (2,1) を結ぶ線分を定義するコマンドである. この線分のように `\path` で定義されるものを「パス」と呼ぶ. 但し, パスは定義しただけでは描画されない. 従って

```
\begin{tikzpicture}
\path (0,0) --(2,1);
\end{tikzpicture}
```

と tex ファイル上に書いても何も表示されない. パスを描画したい場合には `\path` の後に `[draw]` と書く. つまりこの場合は `\path[draw] (0,0) --(2,1);` となる. 更に `\path[draw]` の略記として `\draw` と書くこともできるため `\draw (0,0) --(2,1);` と書くことができる. 以上を踏まえて

```
\begin{tikzpicture}
\draw (0,0) --(2,1);
\end{tikzpicture}
```

と tex ファイル上に書いてコンパイルすると, 次のような図が表示されることになる.



`tikzpicture` 環境は本文中にも数式中にも書くことができるため,

```
\[
\begin{tikzpicture}
\draw (0,0) --(2,1);
\end{tikzpicture}
\]
```

のように `\[ \]` の中に書けば上のように表示されるし,

```
このように本文
\begin{tikzpicture}
\draw (0,0) --(2,1);
\end{tikzpicture}
中表示される
```

のように本文中に書けば, このように本文中表示される<sup>\*3</sup>.

---

<sup>\*3</sup> この例のように, 通常は `tikzpicture` 環境の一番下の部分が本文の baseline に来るように表示される. この挙動を変更するには第 1.7 節の `baseline` オプションや, 第 2.3 節の `use as bounding box` オプションを使用する.

### 1.1.2 略記法 ([1]12.2.2)

`tikzpicture` 環境には簡単に使用するための略記法が用意されている (セミコロンの有無に注意).

- `\tikz{コマンド}`

これは `\begin{tikzpicture}コマンド;\end{tikzpicture}` の略記である.

- `\tikz コマンド;`

これは `\begin{tikzpicture}コマンド;\end{tikzpicture}` の略記である.

故に先に使用した例

```
\begin{tikzpicture}
\draw (0,0) --(2,1);
\end{tikzpicture}
```

であれば `\tikz{\draw (0,0) --(2,1)}` や `\tikz \draw (0,0) --(2,1);` と書くことができる.

### 1.1.3 オプション引数について ([1]12.4.1)

TikZ では、各所でオプション引数を使って様々な設定を行う. 通常の  $\text{\TeX}$  と同様、これは `[ ]` の中にカンマ区切りで記述する. 記述は `オプション名=設定値` の形式で行う. 例えば `\path` に対して「オプション `draw` に `red` を設定し、`fill` に `blue` を設定する」場合は `\path[draw=red,fill=blue]` となる. ここで注意として、設定値に `,` や `]` が含まれる場合には、それを明示するために設定値全体を `{ }` で囲う必要がある<sup>\*4</sup>. 例えばオプション `shift` に `(1,1)` を設定する場合 `\path[shift={(1,1)}]` としなければならない.

もう一つ注意として、オプションには初期値と既定値が与えられている場合がある.

- 初期値 (initial value) とは、最初から設定されている値であり、例えば `\path` のオプション `line width` の初期値は `0.4pt` である. よってこの場合、単に `\path` と書くのは `\path[line width=0.4pt]` と書くのと同じである.
- 既定値 (default value) が与えられているオプションの場合、`=設定値` の部分を省略することができ、省略した場合は既定値が使われる. 例えば `\path` のオプション `rounded corners` の既定値は `4pt` である. よって `\path[rounded corners]` と書くことができ、これは `\path[rounded corners=4pt]` と同じである.

---

<sup>\*4</sup> これは TikZ というよりは  $\text{\TeX}$  の仕様と思われる.



なお、このオプション引数の処理は pgfkeys というパッケージによるものである。詳しくは [1]87 を参照。

## 1.2 座標系

TikZ では「座標」を指定することで様々なものを描いていくことになる。「座標」の指定形式には様々なものがあり、(座標系名 cs: 設定) の書式で指定することができる<sup>\*5</sup>。座標系名 の部分には、以下で述べる canvas 等のような名称を指定する。設定 の部分は第 1.1.3 節で説明したのと同様の形式で、設定する内容をカンマ区切りで設定する。座標系名 には以下のようなものがある。

- canvas ([1]13.2.1)

所謂  $xy$  座標であり、(canvas cs:x=1cm,y=2cm) のように  $x$  軸方向と  $y$  軸方向の長さを設定する。この場合は原点から  $x$  軸方向に 1 cm,  $y$  軸方向に 2 cm の位置を表す。ただこれはよく使う座標系なので、カンマを使って (1cm,2cm) のように書くこともできる。また更に単位を省略して (1,2) と書くこともできる。この場合単位はデフォルトでは cm になる ([1]11.1) <sup>\*6</sup>。更に長さを設定する際に計算式を書くこともできる。例えば (2cm+5pt,-1.5\*1.5mm) のような記述ができる (詳しくは第 2.11 節を参照)。

- canvas polar ([1]13.2.1)

これは所謂極座標で、(canvas polar cs:angle=30,radius=1cm) のように書く。これは角度 30 度、距離 1 cm の点を表す。angle には角度を度数法で設定する。またこの座標系も : を使って (30:1cm) のように略記することができる。また角度に弧度法を使用したい場合は r を使って (2r:1cm) のように書くことができる。これは角度 2 ラジアン、距離 1 cm の点を表す。また円周率は pi と書けるため (pi/2 r:1cm) のような指定の仕方もできる (詳しくは第 2.11 節を参照)。

- node ([1]13.2.3)

これは「名前」を使用して別の点を参照する方法である。詳しくは第 1.5.4 節で述べる。

- xyz ([1]13.2.1)

所謂  $xyz$  座標であり、canvas の 3 次元バージョンである。デフォルトの状態では

---

<sup>\*5</sup> cs は Coordinate System の略。

<sup>\*6</sup> この動作は第 1.4 節で説明する x オプションと y オプションにより変更可能である。

は、 $z$  軸方向の単位ベクトルを  $xy$  平面上の  $(-3.85\text{ mm}, -3.85\text{ mm})$  として描くことで 3 次元空間を描く (第 1.9 節の例を参照). これも `canvas` と同様に  $(1,2,3)$  のような略記ができる.

- `tangent` ([1]13.2.4)  
1 点から円に接線を引いたときの接点の座標を参照することができる. 詳しくは第 2.9 節で述べる.
- `perpendicular` ([1]13.3.1)  
これは別の点から新たな座標を計算する方法である. 詳しくは第 2.7 節で述べる.
- `xyz polar` ([1]13.2.1)  
これも極座標である. 詳しくは第 2.4.1 節で述べる.
- `xy polar` ([1]13.2.1)  
これは `xyz polar` のエイリアスであり, `xyz polar` と完全に同じである.
- `barycentric` ([1]13.2.2)  
これは重心座標系である. 詳しくは [1]13.2.2 を参照.

## 1.3 基本的なオペレーション

ここでは `\path` の中で使用するオペレーションのうち基本的なものについて説明する.

### 1.3.1 Move-To オペレーション ([1]14.1)

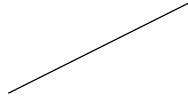
`\path` の内部で単に (座標) と書いた場合 (「座標」の指定方法は第 1.2 節で述べた通り), これは Move-To オペレーションといい「現在位置」を指定した座標に変更する効果を持つ (「現在位置」の意味は次の節以降を読むことで分かるであろう).

先程使用した `\tikz \draw (0,0) --(2,1);` の場合は  $(0,0)$  が Move-To オペレーションである.

### 1.3.2 Line-To オペレーション ([1]14.2)

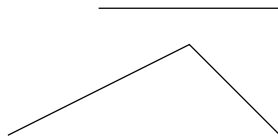
`--(座標)` を Line-To オペレーションといい, 「現在位置」から (座標) までの線分を表す. 例として再び `\tikz \draw (0,0) --(2,1);` を考えると, この場合は `--(2,1)` が Line-To オペレーションである. よってこの例では `\path` の中に 2 つのオペレーションが入っていることになる. このような場合, オペレーションは前から順に処理される. 故にこの例では, まず Move-To オペレーション  $(0,0)$  で  $(0,0)$  に移動した後, Line-To オペレーション `--(2,1)` で,  $(0,0)$  から  $(2,1)$  への線分を描くという動作になるため,

結果として次のような図が描かれる。



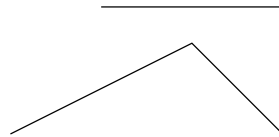
今の例は単なる線分であるが、Line-To オペレーションは「現在位置」も移動するため、例えば次のようにすることで折れ線を描くことができる。

```
\tikz \draw (0,0) --(2,1) --(3,0) --(3,1.4) --(1,1.4);
```



なお通常の  $\text{T}_{\text{E}}\text{X}$  と同様、改行や半角スペースは途中で自由に使えるため次のようにしても同じ絵が描かれる。

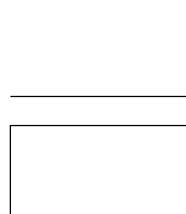
```
\tikz \draw (0,0) -- (2,1) -- (3,0)
-- (3,1.4) -- (1,1.4);
```



-- の他にも -| や |- という指定方法も存在する。-| は点を横線 → 縦線の順で使って繋げる。逆に |- は縦線 → 横線の順となる。

```
\tikz \draw (0,0) -|(2,1);
```

```
\tikz \draw (0,0) |-(2,1);
```



### 1.3.3 rectangle オペレーション ([1]14.4)

rectangle(座標) で「現在位置」と(座標)を頂点とする長方形を表す。例えば次のようになる。

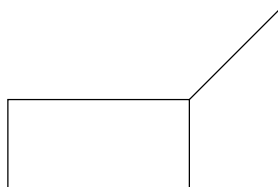
```
\tikz \draw (0,0) rectangle(2,1);
```



また rectangle を -- と組み合わせて使うこともできる (組み合わせて使えるのは、以下で述べる他のオペレーションについても同様である)。例えば

```
\tikz \draw (0,0) rectangle(2,1) --(3,2);
```

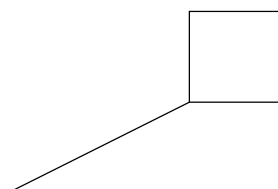
とすると「(0,0) へ移動し」「(2,1) を頂点とする長方形を描き」「(3,2) へと線分を描く」というようになるため次のようになる。



一方で

```
\tikz \draw (0,0) --(2,1) rectangle(3,2);
```

とした場合は「(0,0) へ移動し」「(2,1) へと線分を描き」「(3,2) を頂点とする長方形を描く」となるので次のようになる。



Move-To オペレーションも組み合わせれば次のようなことも可能である (これは「(0,0) へ移動し」「(0,1) へと線分を描き」「(0.5,0) へ移動し」「(1,1) を頂点とする長方形を描く」となる)。

```
\tikz \draw (0,0) --(0,1) (0.5,0) rectangle(1,1);
```



もちろんこれは、\draw を 2 つ使って次のように書いてもよい。

```
\begin{tikzpicture}
\draw (0,0) --(0,1);
\draw (0.5,0) rectangle(1,1);
\end{tikzpicture}
```



### 1.3.4 circle オペレーション ([1]14.6)

circle で「現在位置」を中心とする楕円を表す。楕円の大きさは circle のオプション [x radius=長さ] と [y radius=長さ] で設定する。x 方向と y 方向が同じ長さのときは [radius=長さ] も使用できる。例えば次のようになる。

```
\tikz \draw (0,0) circle[radius=0.5]
(2,0) circle[x radius=1,y radius=0.5];
```

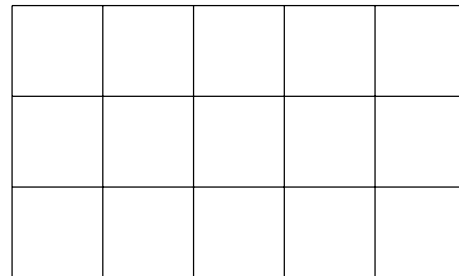


なお `\draw (0,0) circle(0.5);` や `\draw (0,0) ellipse(1 and 0.5);` のような書き方もできるが、これは古い書き方であり避けた方がよいと思われる。

### 1.3.5 grid オペレーション ([1]14.8)

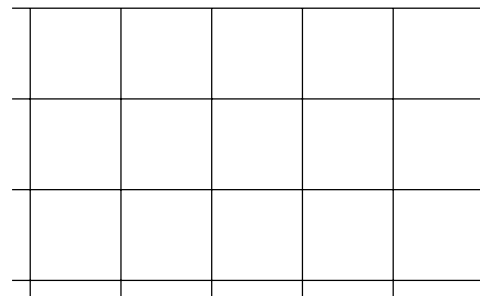
`grid(座標)` を `grid` オペレーションといい、「現在位置」と (座標) を頂点とする長方形の範囲に格子線を描く。例えば次のようになる。

```
\tikz \draw (0,0) grid(5,3);
```



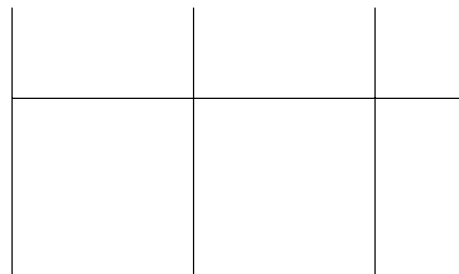
また格子は (0,0) を基準として描かれる (次の例を参照)。

```
\tikz \draw (-0.2,-0.2) grid(5,3);
```



格子の間隔は `grid` のオプション `[step=値]` で設定することができる。値 には 長さ か (座標) を設定する。 `[step=長さ]` と設定した場合は、格子の間隔がその長さになる。これは初期値が 1cm なので、何も設定しなければ格子の間隔は 1cm になる。単位は省略可能で、その場合の単位はデフォルトでは cm になる<sup>\*7</sup>。

```
\tikz \draw (0,0) grid[step=2](5,3);
```

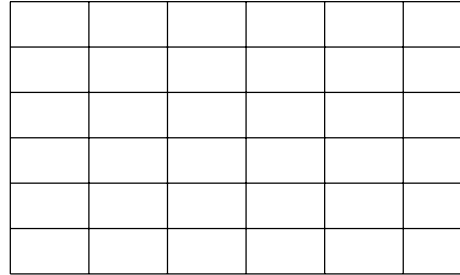


---

<sup>\*7</sup> この動作も第 1.4 節で説明する `x` オプションと `y` オプションにより変更される。

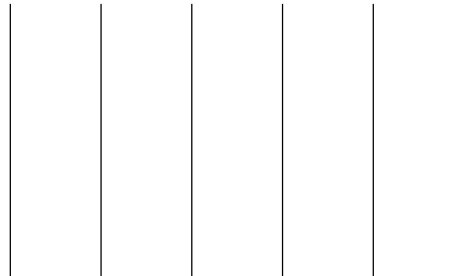
[step=(座標)] と指定した場合は、(座標) の  $x$  軸方向の長さと  $y$  軸方向の長さをそれぞれ使用する。

```
\tikz \draw (0,0)
  grid[step=(30:1cm)] (5,3);
```



なお間隔が 0 になった場合は線が引かれなくなる。

```
\tikz \draw (0,0)
  grid[step={(1,0)}] (5,3);
```

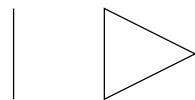


$x$  軸方向の間隔のみを設定したい場合は [xstep=長さ] を使うことができる。同様に  $y$  軸方向については [ystep=長さ] を使用することができる。

### 1.3.6 cycle ([1]14.2.1)

(座標) の代わりに cycle と書くことができる。これは直前に使用した Move-To オペレーションの座標を表す。例えば次の例のようになる。

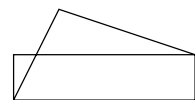
```
\tikz \draw (0,0) --(0,1) (1,0) --(1,1) --(2,0.5) --cycle;
```



```
\tikz \draw (0,0) --(1,1) -|cycle;
```



```
\tikz \draw (0,0) --(0.5,1) --(2,0.5) rectangle cycle;
```



## 1.4 パスのオプション

この節では `\path` に付けられるオプションについて説明する.

- `[draw=色]` ([1]15.3)

指定した色でパスを描く. TikZ における色の指定では `xcolor` と同じものが使用できる (詳しくは第 2.2 節を参照).

```
\tikz \path[draw=red] (0,0) --(1,0);
```



また `[draw=none]` と設定するとパスは描かれなくなる. 既定値は通常は `black` であり, 単に `[draw]` と書いた場合は, 今まで見てきた通りパスが黒で描かれる. ここで「通常は」と書いたのは, `draw` オプションは `tikzpicture` 環境に対して付けると, 既定値を変える効果を持つからである (次の例を参照).

```
\begin{tikzpicture}[draw=red]
\path[draw]      (0,2) --(1,2);
\path            (0,1) --(1,1);
\path[draw=black] (0,0) --(1,0);
\end{tikzpicture}
```



- `[line width=長さ]` ([1]15.3.1)

パスの太さを設定する. 初期値は `0.4pt` である.

```
\tikz \draw[line width=5pt] (0,0) --(1,0);
```



ちなみにこれを使えば, 次の 2 つの書き方には違いがあることが分かる.

```
\begin{tikzpicture}
\draw[line width=5pt]
  (0,0) --(1,0) --(1,1);
\draw[line width=5pt]
  (1.5,0) --(2.5,0) (2.5,0) --(2.5,1);
\end{tikzpicture}
```



`cycle` についても次のような違いがある.

```
\begin{tikzpicture}
\draw[line width=5pt]
  (0,0) --(1,0) --(0,1) --(0,0);
\draw[line width=5pt]
  (1.5,0) --(2.5,0) --(1.5,1) --cycle;
\end{tikzpicture}
```



- `[ultra thin]` ([1]15.3.1)

`[line width=0.1pt]` と同じ.

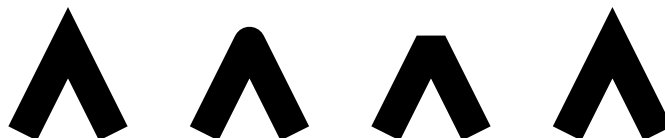
- `[very thin]` ([1]15.3.1)  
`[line width=0.2pt]` と同じ.
- `[thin]` ([1]15.3.1)  
`[line width=0.4pt]` と同じ.
- `[semithick]` ([1]15.3.1)  
`[line width=0.6pt]` と同じ.
- `[thick]` ([1]15.3.1)  
`[line width=0.8pt]` と同じ.
- `[very thick]` ([1]15.3.1)  
`[line width=1.2pt]` と同じ.
- `[ultra thick]` ([1]15.3.1)  
`[line width=1.6pt]` と同じ.
- `[line cap=種類]` ([1]15.3.1)  
パスの先端の部分の形状を設定する. 種類 には `round`, `rect`, `butt` の 3 つを設定できる. `butt` が初期値である. 設定時に具体的にどうなるかは次の例を参照.

```
\begin{tikzpicture}
\draw[line width=10pt] (0,1.5) --(1,1.5);
\draw[line width=10pt,line cap=round] (0,1) --(1,1);
\draw[line width=10pt,line cap=rect] (0,0.5) --(1,0.5);
\draw[line width=10pt,line cap=butt] (0,0) --(1,0);
\end{tikzpicture}
```



- `[line join=種類]` ([1]15.3.1)  
折れ線部分の形状を設定する. 種類 は `round`, `bevel`, `miter` の 3 つを設定できる. `miter` が初期値である. 設定時に具体的にどうなるかは次の例を参照.

```
\begin{tikzpicture}
\draw[line width=10pt] (0,0) --(0.5,1) --(1,0);
\draw[line width=10pt,line join=round] (2,0) --(2.5,1) --(3,0);
\draw[line width=10pt,line join=bevel] (4,0) --(4.5,1) --(5,0);
\draw[line width=10pt,line join=miter] (6,0) --(6.5,1) --(7,0);
\end{tikzpicture}
```

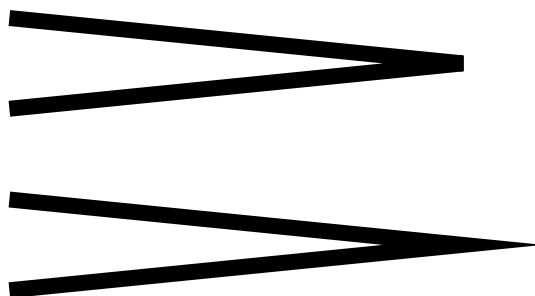


- `[miter limit=倍率]` ([1]15.3.1)  
`[line join=miter]` の場合, 先端がある程度鋭くなると自動的に `bevel` の形状



になる。どのくらい鋭くなったら bevel になるかを指定するのが [miter limit] である。倍率 には 1 以上の値を指定し、line width の何倍の距離を離れたら bevel にするかを表す。[miter limit] の初期値は 10 である。

```
\begin{tikzpicture}
\draw[line width=5pt] (0,2) --(5,2.5) --(0,3);
\draw[line width=5pt,miter limit=15] (0,0) --(5,0.5) --(0,1);
\end{tikzpicture}
```



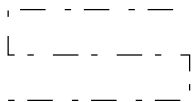
- [dash pattern=パターン] ([1]15.3.2)

点線などを描くためのオプションである。パターン には、例えば

on 2pt off 4pt on 8pt off 6pt

のような記述をする。これは「2pt 描き 4pt 空け、8pt 描き 6pt 空け」を繰り返す「パターン」である。

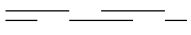
```
\tikz \draw[dash pattern=on 2pt off 4pt on 8pt off 6pt]
(0,0) --(2,0) --(2,0.5) --(0,0.5) --(0,1) --(2,1);
```



- [dash phase=長さ] ([1]15.3.2)

[dash pattern] の開始位置を 長さ の分だけずらす。初期値は 0pt である。


```
\begin{tikzpicture}
\draw[dash pattern=on 20pt off 10pt]
(0,3pt) --(2,3pt);
\draw[dash pattern=on 20pt off 10pt,dash phase=10pt]
(0,0) --(2,0);
\end{tikzpicture}
```



- [dash=パターン phase 長さ] ([1]15.3.2)

[dash pattern] と [dash phase] をまとめて設定することができる。

```
\begin{tikzpicture}
\draw[dash=on 20pt off 10pt phase 10pt]
(0,0) --(2,0);
\end{tikzpicture}
```

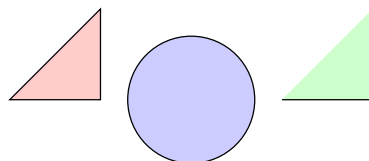


- [solid] ([1]15.3.2)  
[dash pattern=] と同じ\*8.
- [dotted] ([1]15.3.2)  
[dash pattern=on \pgflinewidth off 2pt] と同じ.
- [densely dotted] ([1]15.3.2)  
[dash pattern=on \pgflinewidth off 1pt] と同じ.
- [loosely dotted] ([1]15.3.2)  
[dash pattern=on \pgflinewidth off 4pt] と同じ.
- [dashed] ([1]15.3.2)  
[dash pattern=on 3pt off 3pt] と同じ.
- [densely dashed] ([1]15.3.2)  
[dash pattern=on 3pt off 2pt] と同じ.
- [loosely dashed] ([1]15.3.2)  
[dash pattern=on 3pt off 6pt] と同じ.
- [dash dot] ([1]15.3.2)  
[dash pattern=on 3pt off 2pt on \the\pgflinewidth off 2pt] と同じ.
- [densely dash dot] ([1]15.3.2)  
[dash pattern=on 3pt off 1pt on \the\pgflinewidth off 1pt] と同じ.
- [loosely dash dot] ([1]15.3.2)  
[dash pattern=on 3pt off 4pt on \the\pgflinewidth off 4pt] と同じ.
- [fill=色] ([1]15.5)  
指定した色でパスの「内側」を塗りつぶす (「内側」の意味は以下の例を参照, 詳しくは [1]15.5.2).

```

\begin{tikzpicture}
\draw[fill=red!20] (0,0) --(1,0) --(1,1) --cycle;
\draw[fill=blue!20] (2,0) circle[radius=0.7];
\draw[fill=green!20] (3,0) --(4,0) --(4,1);
\end{tikzpicture}

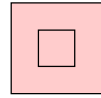
```



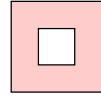

---

\*8 これらは `texmf-dist\tex\generic\pgf\frontendlayer\tikz\tikz.code.tex` で定義されている.

```
\begin{tikzpicture}
\draw[fill=red!20] (0,0) --(0,1) --(1,1) --(1,0) --cycle
(0.3,0.3) --(0.3,0.7) --(0.7,0.7) --(0.7,0.3) --cycle;
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw[fill=red!20] (0,0) --(0,1) --(1,1) --(1,0) --cycle
(0.3,0.3) --(0.7,0.3) --(0.7,0.7) --(0.3,0.7) --cycle;
\end{tikzpicture}
```



なお `\path[fill]` の略記として `\fill` を, `\path[fill,draw]` の略記として `\filldraw` を使う事ができる. ちなみに `fill` と `draw` については, 順番としてはまず `fill` で塗りつぶされた後に `draw` で線が描かれることになる. 次の例を参照.

```
\tikz \draw[fill=red!20,line width=5pt]
(0,0) --(1,0) --(1,1);
```



また `[fill=none]` と設定すると塗りつぶしはされなくなる.

- `[color=色]` ([1]15.2)

`[draw]` や `[fill]` について設定した 色 を使用する (後述する `[node contents]` に対してもこの 色 は適用される).

```
\tikz \draw[color=red] (0,0) --(1,0);
```



```
\tikz \fill[color=red] (0,1) --(1,1) --(1,1.5) --cycle;
```



`color=` は省略できる.

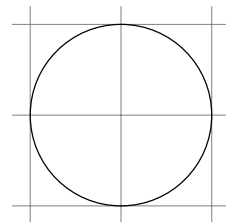
```
\tikz \draw[red] (0,0) --(1,0);
```



- `[help lines]`

`[color=gray,line width=0.2pt]` と同じ. これは主に `grid` オペレーションで使用する.

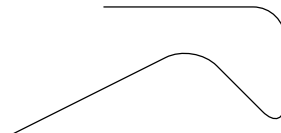
```
\begin{tikzpicture}
\draw[help lines] (-1.2,-1.2) grid(1.2,1.2);
\draw (0,0) circle[radius=1cm];
\end{tikzpicture}
```



- [rounded corners=長さ] ([1]14.5)

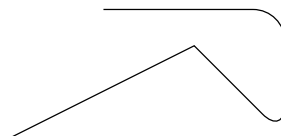
角を丸くしたい場合に使用する．長さ には「丸」の大きさを設定する．4pt が既定値である．

```
\tikz \draw[rounded corners=10pt] (0,0) --(2,1)
      --(3,0) --(3,1.4) --(1,1.4);
```



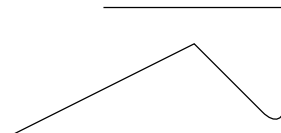
今の例は全ての角が丸くなっているが，次のようにオプションを途中に書くと，その場所以降の角のみ丸くなる．

```
\tikz \draw (0,0) --(2,1)
      [rounded corners=10pt] --(3,0) --(3,1.4)
      --(1,1.4);
```



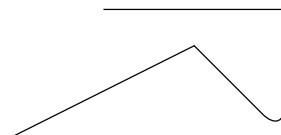
更に途中から元に戻したい場合は [sharp corners] を指定する．

```
\tikz \draw (0,0) --(2,1)
      [rounded corners=10pt] -- (3,0)
      [sharp corners] --(3,1.4) --(1,1.4);
```



もしくは、{ } を使って範囲を指定することもできる ([1]12.3.4)．

```
\tikz \draw (0,0) --(2,1)
      {[rounded corners=10pt] --(3,0)} --(3,1.4)
      --(1,1.4);
```



[rounded corners] は rectangle に対しても有効である．

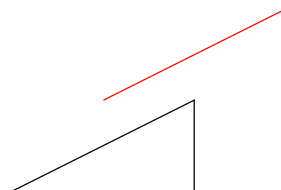
```
\tikz \draw[rounded corners] (0,0) rectangle(2,1);
```



- [shift=(座標)] ([1]25.3)

設定した (座標) の分だけ \path に現れる「座標」を平行移動する．例えば次のように [shift={(1,1)}] と書いた場合<sup>\*9</sup>， $x$  軸方向に 1， $y$  軸方向に 1 移動する．

```
\begin{tikzpicture}
\draw (0,0) --(2,1) --(2,0);
\draw[red,shift={(1,1)}] (0,0) --(2,1) --(2,0);
\end{tikzpicture}
```

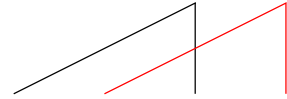


<sup>\*9</sup> 第 1.1.3 節で注意した通り，カンマを使う場合 (つまり座標系名 `canvas` を使う場合) は { } で囲わなければならない．

- [xshift=長さ] ([1]25.3)

$x$  軸方向に平行移動する.

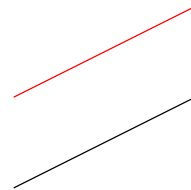
```
\begin{tikzpicture}
\draw (0,0) --(2,1) --(2,0);
\draw[red,xshift=1cm] (0,0) --(2,1) --(2,0);
\end{tikzpicture}
```



- [yshift=長さ] ([1]25.3)

$y$  軸方向に平行移動する.

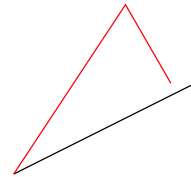
```
\begin{tikzpicture}
\draw (0,0) --(2,1) --(2,0);
\draw[red,yshift=1cm] (0,0) --(2,1) --(2,0);
\end{tikzpicture}
```



- [rotate=角度] ([1]25.3)

原点を中心に, 設定した 角度 だけ, \path に現れる「座標」を回転させる.

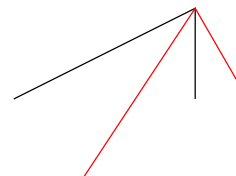
```
\begin{tikzpicture}
\draw (0,0) --(2,1) --(2,0);
\draw[red,rotate=30] (0,0) --(2,1) --(2,0);
\end{tikzpicture}
```



- [rotate around=角度:(座標)] ([1]25.3)

設定した (座標) を中心に, 設定した 角度 だけ, \path に現れる「座標」を回転させる.

```
\begin{tikzpicture}
\draw (0,0) --(2,1) --(2,0);
\draw[red,rotate around={30:(2,1)}]
(0,0) --(2,1) --(2,0);
\end{tikzpicture}
```



- [scale=倍率] ([1]25.3)

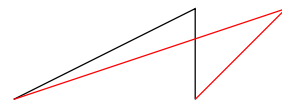
原点を中心に, 設定した 倍率 だけ, \path に現れる「座標」を拡大させる. 負の値も設定可能である. 倍率 の絶対値は過度に大きくしたり小さくしたりすべきではないとのこと. 「座標」を拡大するだけなので, 例えば線の太さなどは拡大されない.

```
\begin{tikzpicture}
\draw[line width=5pt] (0,1) --(1,1);
\draw[line width=5pt,scale=2] (0,0) --(1,0);
\end{tikzpicture}
```



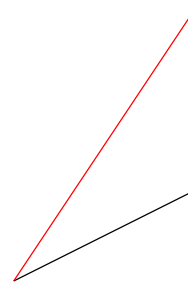
- `[scale around=倍率:(座標)]` ([1]25.3)  
設定した (座標) を中心に, 設定した 倍率 だけ, 「座標」を拡大させる.
- `[xscale=倍率]` ([1]25.3)  
 $x$  軸方向に拡大する.
- `[yscale=倍率]` ([1]25.3)  
 $y$  軸方向に拡大する.
- `[xslant=数値]` ([1]25.3)  
 $x$  軸方向に 数値 の分だけ傾ける.

```
\begin{tikzpicture}
\draw (0,0) --(2,1) --(2,0);
\draw[red,xslant=1] (0,0) --(2,1) --(2,0);
\end{tikzpicture}
```



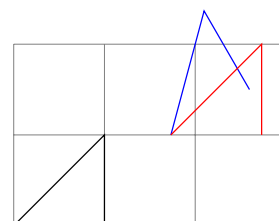
- `[yslant=数値]` ([1]25.3)  
 $y$  軸方向に 数値 の分だけ傾ける.

```
\begin{tikzpicture}
\draw (0,0) --(2,1) --(2,0);
\draw[red,yslant=1] (0,0) --(2,1) --(2,0);
\end{tikzpicture}
```



- `[shift only]` ([1]25.3)  
これは次のような動作になる. 設定されているオプションによる (0,0) の行き先を (p) とする. このとき `[shift only]` は `[shift=(p)]` と同等である. 次の例を参照.

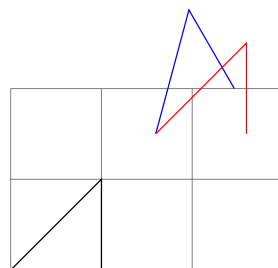
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[rotate=30,xshift=2cm,blue]
(0,0) -- (1,1) -- (1,0);
\draw[rotate=30,xshift=2cm,red,shift only]
(0,0) -- (1,1) -- (1,0);
\end{tikzpicture}
```



```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (1,0);
\draw[rotate around={30:(-1,0)},xshift=2cm,
blue] (0,0) -- (1,1) -- (1,0);
\draw[rotate around={30:(-1,0)},xshift=2cm,
red,shift only]
(0,0) -- (1,1) -- (1,0);
\end{tikzpicture}

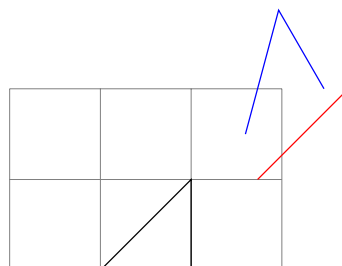
```



```

\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (1,0) -- (2,1) -- (2,0);
\draw[rotate=30,xshift=2cm,blue]
(1,0) -- (2,1) -- (2,0);
\draw[rotate=30,xshift=2cm,red,
shift only]
(1,0) -- (2,1) -- (2,0);
\end{tikzpicture}

```



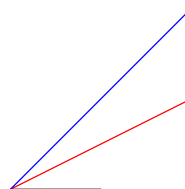
- [x=(座標)] ([1]25.2)

座標系名 `canvas` や `grid` オペレーションなどで使用する,  $x$  軸方向の単位ベクトルを変更する. 次の例を参照.

```

\begin{tikzpicture}
\draw (0,0) --(1,0);
\draw[x={(2,1)},red] (0,0) --(1,0);
\draw[x={(2,1)},blue] (0,0) --(1,1);
\end{tikzpicture}

```



このオプションは [x=長さ] という形式での設定も可能である. この方法で設定するのは [x={(長さ,0pt)}] と設定するのと同じ意味である. またこのオプションの初期値は [x=1cm] である.

- [y=(座標)] ([1]25.2)

[x=(座標)] と同様に,  $y$  軸方向の単位ベクトルを変更する. このオプションも [y=長さ] という形式での設定も可能であり, [y={(0pt,長さ)}] と同じ意味である. またこのオプションの初期値は [y=1cm] である.

- [z=(座標)] ([1]25.2)


座標系名 `xyz` などで使用する,  $z$  軸方向の単位ベクトルを変更する. このオプションも [z=長さ] という形式での設定も可能であり, [z={(長さ,長さ)}] と同じ意味である. またこのオプションの初期値は [z=-3.85mm] である.

- [transform canvas=設定] ([1]25.4)

\path に 設定 を適用する. 設定 には `shift={(1,1)}` や `scale=2` のようなもの

を書く．これらの 設定 を直接 `\path` のオプションに書くのとは微妙に処理が異なる．例えば次の例では線の太さにも `[scale=2]` が適用されている．第 1.9 節も参照．

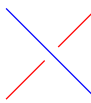
```
\begin{tikzpicture}
\draw[line width=5pt] (0,1) --(1,1);
\draw[line width=5pt,
      transform canvas={scale=2}] (0,0) --(1,0);
\end{tikzpicture}
```



- `[preaction=設定]` ([1]15.10)

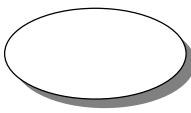
`\path` を実行する前に 設定 に書いた内容で `\path` を実行する．例えば次のように `[preaction={draw=white,line width=6pt}]` と書くと (0,1) から (1,0) へ、白い太線を描いた後で青線を描くので、青線が赤線の上に立体的に交差しているようになる．

```
\begin{tikzpicture}
\draw[red] (0,0) --(1,1);
\draw[preaction={draw=white,line width=6pt},blue]
      (0,1) --(1,0);
\end{tikzpicture}
```



他の使い方としては `[preaction]` を使うことで影を付けることができる．

```
\begin{tikzpicture}
\draw[fill=white,preaction={fill=gray,
      transform canvas={shift={(1mm,-1mm)}}}]
      (0,0) circle[x radius=1cm,y radius=5mm];
\end{tikzpicture}
```



これをより一般的に行うために `\usetikzlibrary{shadows}` というものもある．詳しくは [1]70 を参照．

- `[postaction=設定]` ([1]15.10)

`[preaction]` と同様であり、`\path` 本体が実行された後に 設定 の内容が実行される．

- `[arrows=開始部分-終了部分]`

パスの形状を設定する．詳しくは第 1.6 節で説明する．

- `[draw opacity=割合]` ([1]23.2)

パスの不透明度を設定する．割合 には 0 ～ 1 を設定することができ、1 の場合が



不透明で 0 は完全な透明である.

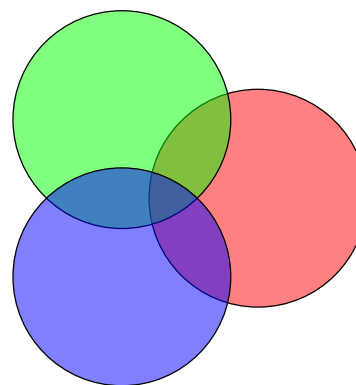
```
\begin{tikzpicture}
\draw[line width=10pt,red] (0.5,0) --(0.5,2);
\draw[line width=10pt,draw opacity=0] (0,2) --(1,2);
\draw[line width=10pt,draw opacity=0.5] (0,1) --(1,1);
\draw[line width=10pt,draw opacity=1] (0,0) --(1,0);
\end{tikzpicture}
```



- [fill opacity=割合] ([1]23.2)

[fill] の不透明度を設定する. 割合 には 0 ~ 1 を設定することができる.

```
\begin{tikzpicture}
\filldraw[fill opacity=0.5,fill=red]
( 0:1cm) circle (12mm);
\filldraw[fill opacity=0.5,fill=green]
(120:1cm) circle (12mm);
\filldraw[fill opacity=0.5,fill=blue]
(-120:1cm) circle (12mm);
\end{tikzpicture}
```



- [opacity=割合] ([1]23.2)  
[draw opacity] と [fill opacity] に同じ 割合 を設定する.
- [ultra nearly transparent] ([1]23.2)  
[opacity=0.05] と同じ.
- [very nearly transparent] ([1]23.2)  
[opacity=0.1] と同じ.
- [nearly transparent] ([1]23.2)  
[opacity=0.25] と同じ.
- [semitransparent] ([1]23.2)  
[opacity=0.5] と同じ.
- [nearly opaque] ([1]23.2)  
[opacity=0.75] と同じ.
- [very nearly opaque] ([1]23.2)  
[opacity=0.9] と同じ.
- [ultra nearly opaque] ([1]23.2)  
[opacity=0.95] と同じ.

- [opaque] ([1]23.2)  
[opacity=1] と同じ.
- [shade] ([1]15.7)  
シェーディングを使ってパスの内側を塗りつぶす. [fill] と同時に設定した場合は [shade] が優先される (ので意味がない). \path[shade] の略記として \shade が, \path[shade,draw] の略記として \shadedraw ができる.

```
\tikz \shade (0,0) rectangle(1,1);
```



- [shading=シェーディング名] ([1]15.7)  
どのような形式でシェーディングするか名前を設定する. シェーディング名 には axis, ball, radial が使用できる他, ライブラリを読み込むことで他にも色々使用できる. 詳しくは [1]69 を参照.

```
\tikz \shade[shading=axis] (0,0) rectangle(1,1);
```



```
\tikz \shade[shading=ball] (0,0) rectangle(1,1);
```



```
\tikz \shade[shading=radial] (0,0) rectangle(1,1);
```



- [top color=色] ([1]69)  
[shading=axis] で上側に使用する色を設定する. このオプションを使用すると, 自動的に [shade,shaping=axis] が設定される.

```
\tikz \path[top color=red] (0,0) rectangle(1,1);
```



- [bottom color=色] ([1]69)  
[top color=色] と同様.
- [middle color=色] ([1]69)  
[shading=axis] で中央に使用する色を設定する. これを [top color] 等の色設定と併用するときは, [middle color] を後に設定する必要がある.

```
\tikz \path[top color=red,bottom color=blue,  
middle color=green] (0,0) rectangle(1,1);
```



- [shading angle=角度] ([1]15.7)  
シェーディングの角度を回転させる.

```
\tikz \shade[shading=axis,shading angle=60]
(0,0) rectangle(1,1);
```



- [left color=色] ([1]69)  
[top color=色,shading angle=90] と同様.

```
\tikz \path[left color=red] (0,0) rectangle(1,1);
```



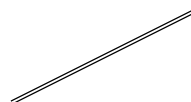
- [right color=色] ([1]69)  
[left color=色] と同様.
- [ball color=色] ([1]69)  
[shading=ball] で使用する色を設定する. このオプションを使用すると, 自動的に [shade,shading=ball] が設定される.

```
\tikz \shade[ball color=red] (0,0) circle[radius=5mm];
```



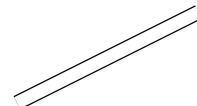
- [double=色] ([1]15.3.4)  
太い線を引いた後で, 色 で指定した色の線を引く. 既定値は white である. よって通常は 2 本線を引いたように見える.

```
\tikz \draw[double] (0,0) --(2,1);
```



- [double distance=長さ] ([1]15.3.4)  
[double=色] を使用したときの 色 で引く線の太さ (つまり 2 本線の間隔) を設定する. 初期値は 0.6pt である. このオプションを設定すると自動で [double] が設定される.

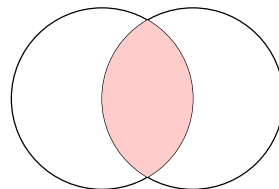
```
\tikz \draw[double distance=4pt] (0,0) --(2,1);
```



- [clip] ([1]15.9)  
\path に [clip] を設定すると, 以降はそのパスの内部だけ描写されるようにな

る。例えば次のようにすると共通部分に色を塗ることができる。

```
\begin{tikzpicture}
\draw      (0,0) circle[radius=1cm];
\draw[clip] (1,0) circle[radius=1cm];
\fill[red!20] (0,0) circle[radius=1cm];
\end{tikzpicture}
```



`\path[clip]` の略記として `\clip` を使うことができる。また `[clip]` の影響範囲を指定したい場合は `scope` 環境 (第 1.8.2 節参照) を使うことができる。

[1]15.9 によると、PDF の仕様上 `[clip]` を設定するパスには適用できないオプションがいくつかあるようなので、あまり複雑な使い方はせず単に `[clip]` は単体で使用するのが良さそうである。

## 1.5 node オペレーション ([1]17.2)

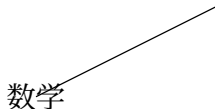
絵の中に文字を書く際に使用するのが「ノード」である。ノードを使うことで、今まで述べてきたパスに対して文字を付けたりすることができる。ノードを作成するのに使用するのが node オペレーションで、これは `{ }` を使って `node{文字}` という形式で書く。これにより「現在位置」に 文字 を描くことができる。例えば

```
\draw (1,2) node{数学};
```

とすると (1,2) の位置に「数学」という文字が書かれる<sup>\*10</sup>。もちろん他の、例えば Line-To オペレーションと組み合わせて使う事もできる。例えば

```
\tikz \draw (0,0) node{数学} --(2,1);
```

とすると、(0,0) と (2,1) を繋ぐ線分を描き、更に (0,0) に「数学」という文字を書くことになる<sup>\*11</sup>。



ちなみに `{文字}` と書く代わりに `node contents=文字` オプションで 文字 を設定することもできる。例えば先の例では

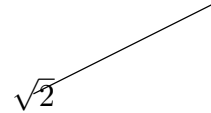
```
\tikz \draw (0,0) node[node contents=数学] --(2,1);
```

<sup>\*10</sup> この文字は `draw` オプションが無くても描かれるため `\path (1,2) node{数学};` としてもよい

<sup>\*11</sup> TikZ の動きとしては、node オペレーションはパスが描かれた後で実行されるので、このような説明になる。

と書くこともできる。なおノードとして数式を書きたい場合は普通の文章と同じように \$ を使って、例えば `node{\sqrt{2}}` のように書けばよい (`node contents` オプションを使う場合は `node[node contents=\sqrt{2}]` となる)。

```
\tikz \draw (0,0) node{\sqrt{2}} --(2,1);
```



### 1.5.1 ノードの形 ([1]17.2.1)

ノードには「形」が存在する。これは `node` に `[draw]` もしくは `[fill]` を指定することで見ることができる (次の例を参照)。

```
\tikz \path (0,0) node[draw]{黒線} (2,0) node[fill=red!20]{赤背景};
```

黒線

赤背景

このようにノードの形は基本的には `rectangle` になっている。形を変えるには `shape` オプションを設定する。通常は次の 3 つが使用できる。

- `[shape=rectangle]`  
これは `shape` オプションの初期値である。
- `[shape=circle]`  
ノードの「形」が円になる。

```
\tikz \path (0,0) node[draw,shape=circle]{三角形};
```

三角形


- `[shape=coordinate]`  
これについては第 1.5.5 節で述べる。

なお `shape=` は省略してもよい。更にライブラリを使うことで他にも様々な「形」を使う事ができる ([1]71 を参照)。また `[draw]`, `[fill]` 以外にも `[shift]`, `[rotate]`, `[scale]`, `[shade]` など、パスのオプションの一部は `node` に対しても使うことができる。

### 1.5.2 at オプション ([1]17.2.1)

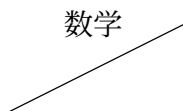
ノードのオプションとして `at` オプションがある。これはノードの「座標」を設定するオプションであり、設定されている場合はこの「座標」が優先して使われる。例えば次の例では、ノードの文字「数学」は (0,0) ではなく (1,1) に書かれる。

```
\tikz \draw (0,0) node[at={(1,1)}]{数学} --(2,1);
```



また `[at=(座標)]` と書く代わりに `node at (座標)` と書くこともできる。

```
\tikz \draw (0,0) node at (1,1) {数学} --(2,1);
```



よって、例えば単に座標 (1,2) にノードを置きたい場合は

```
\path node at (1,2) {数学};
```

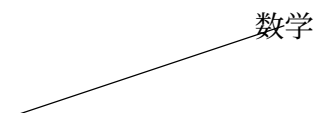
と書くことができる。また `\path node` の略記として `\node` と書くことができる。よって

```
\node at (1,2) {数学};
```

という書き方もできる。

### 1.5.3 pos オプション ([1]17.8)

まず、普通に `\tikz \draw (0,0) --(3,1) node{数学};` とすると (3,1) に「数学」という文字が書かれる。



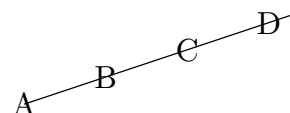
ここで `node` のオプションで `[pos=割合]` という設定ができる。例えば `[pos=0.5]` と設定すると (0,0) と (3,1) の間に「数学」を書くことができる。

```
\tikz \draw (0,0) --(3,1) node[pos=0.5]{数学};
```



割合には 0 ~ 1 を設定することができ、0 が 1 つ前の位置、1 が「現在位置」である。

```
\tikz \draw (0,0) --(3,1)
  node[pos=0]{A} node[pos=0.3]{B}
  node[pos=0.6]{C} node[pos=0.9]{D};
```

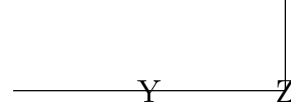


```
\tikz \draw (0,0) --(3,0) --(3,1)
      node[pos=0.5]{X};
```



-| や |- の場合でも [pos] は使用できる. この場合, 角の部分が 0.5 の位置になる.

```
\tikz \draw (0,0) -|(3,1)
      node[pos=0.25]{Y} node[pos=0.5]{Z};
```



\draw (0,0) --(3,1) node[pos=0.5]{数学}; という書き方はかなり不自然な気がするが, 実は同じことを \draw (0,0) --node{数学}(3,1); と書くことができる (この場合 [pos=0.5] が自動設定されている).

```
\tikz \draw (0,0) --node{数学}(3,1);
```



またこの場合でも [pos] は使用できる.

```
\tikz \draw (0,0) --node[pos=0.7]{数学}(3,1);
```



## 1.5.4 ノードに名前を付ける

ノードには「名前」を付けることで, その位置を後から「名前」で参照することができる. 「名前」は node の name オプションで設定する. 例えば

```
\node[name=math] at (1,2) {数学};
```

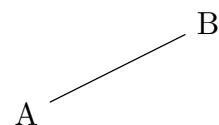
とすると, このノードには math という「名前」が付く. もしくは ( ) を使って node (名前) という形式でも「名前」を付けることができる. 例えば先の例では

```
\node (math) at (1,2) {数学};
```

と書くことになる.

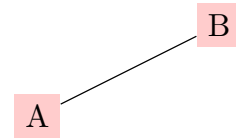
参照するときには, 座標系名 node を使用して (node cs:name=名前) と書く. もしくは省略して単に (名前) と書くこともできる. 例えば次のようにすると A と B の間に線が引かれる.

```
\begin{tikzpicture}
\node (hoge) at (0,0) {A};
\node (fuga) at (2,1) {B};
\draw (node cs:name=hoge) --(node cs:name=fuga);
\end{tikzpicture}
```



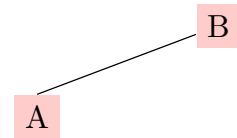
見てなんとなくわかる通り，この場合ノードの「形」に合わせて線が引かれる．次のように `node` に `[fill]` を指定してみると分かる．

```
\begin{tikzpicture}
\node[fill=red!20] (hoge) at (0,0) {A};
\node[fill=red!20] (fuga) at (2,1) {B};
\draw (hoge) --(fuga);
\end{tikzpicture}
```

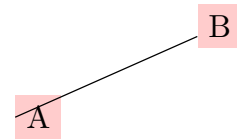


今の場合は自動的に線が引かれたが，始点と終点の「場所」を明示的に指定することもできる．その方法は2つあり，一つは `anchor=位置名` である．

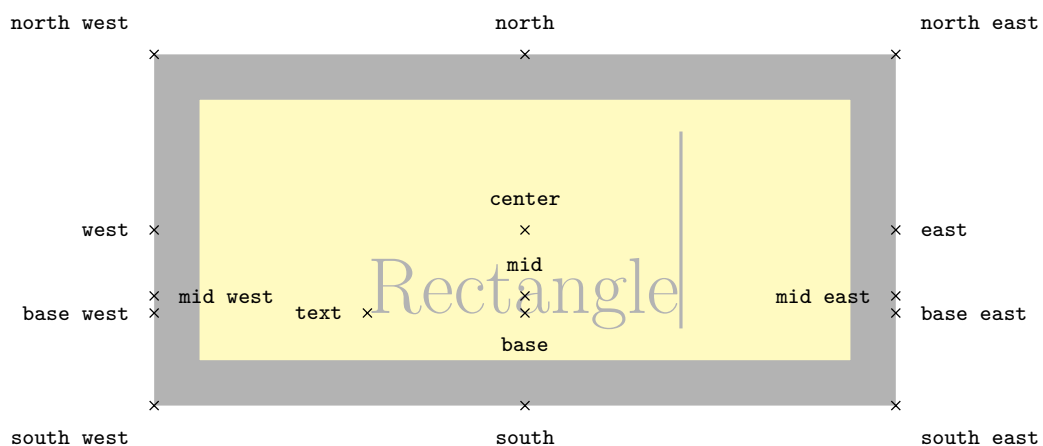
```
\begin{tikzpicture}
\node[fill=red!20] (hoge) at (0,0) {A};
\node[fill=red!20] (fuga) at (2,1) {B};
\draw (node cs:name=hoge,anchor=north)
--(node cs:name=fuga);
\end{tikzpicture}
```



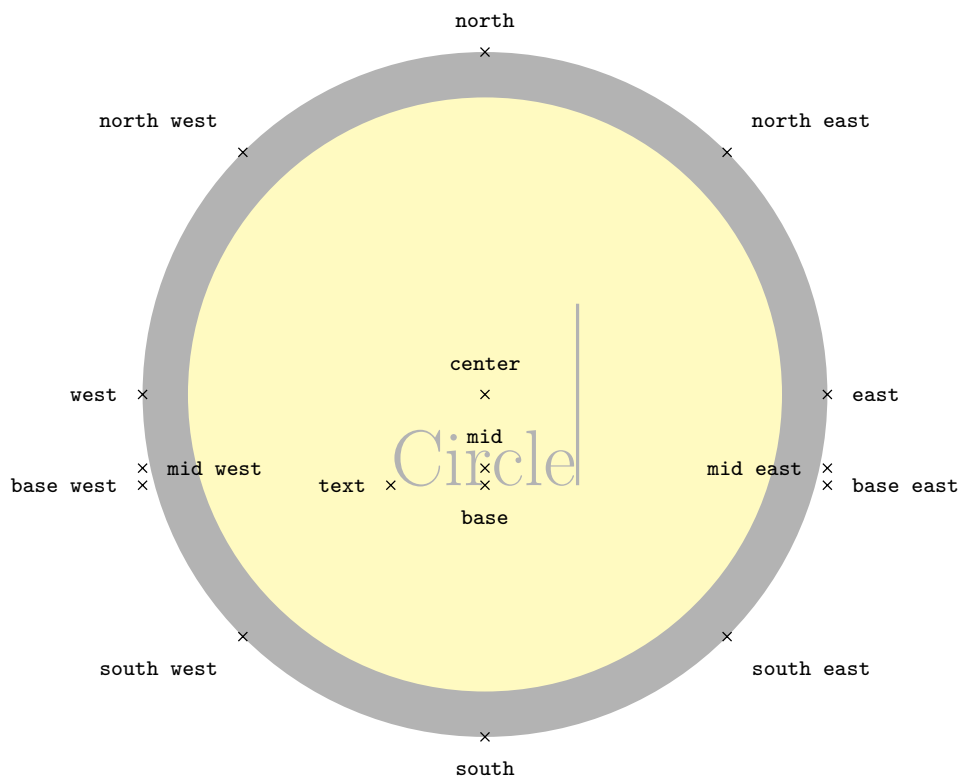
```
\begin{tikzpicture}
\node[fill=red!20] (hoge) at (0,0) {A};
\node[fill=red!20] (fuga) at (2,1) {B};
\draw (node cs:name=hoge,anchor=west)
--(node cs:name=fuga);
\end{tikzpicture}
```



位置名 に使える名称はノードの `shape` オプションの値により異なる．`rectangle` と `circle` の場合，使える名称は以下のとおりである (この図は [1]71.2 のものを利用した)．







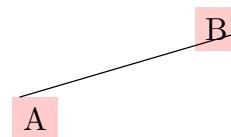
例えばノードの文字に赤い下線を引きたい場合、次のようにするとできる.

```
\begin{tikzpicture}
\node[inner sep=1.5pt] (hoge) at (0,0) {テキスト};
\draw[red,line width=1pt]
  (node cs:name=hoge,anchor=south west)
  --(node cs:name=hoge,anchor=south east);
\end{tikzpicture}
```

テキスト

もう一つの指定方法は `angle=角度` である.

```
\begin{tikzpicture}
\node[fill=red!20] (hoge) at (0,0) {A};
\node[fill=red!20] (fuga) at (2,1) {B};
\draw (node cs:name=hoge,angle=125)
  --(node cs:name=fuga,angle=350);
\end{tikzpicture}
```



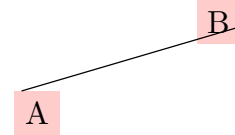
これらにも略記法があり (`node cs:name=名前,anchor=位置名`) は (`名前.位置名`) と書ける.

```
\begin{tikzpicture}
\node[inner sep=1.5pt] (hoge) at (0,0) {テキスト};
\draw[red,line width=1pt]
  (hoge.south west) --(hoge.south east);
\end{tikzpicture}
```

テキスト

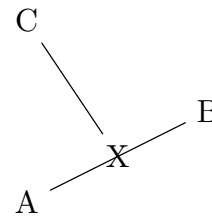
(node cs:name=名前,angle=角度) も同様に (名前. 角度) と書くことができる。

```
\begin{tikzpicture}
\node[fill=red!20] (hoge) at (0,0) {A};
\node[fill=red!20] (fuga) at (2,1) {B};
\draw (hoge.125) --(fuga.350);
\end{tikzpicture}
```



また node であれば ( ) で名前を付けられるので、例えば次のようなこともできる。

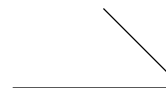
```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (2,1) {B};
\node (c) at (0,2) {C};
\draw (a) -- node (x) {X} (b);
\draw (x) --(c);
\end{tikzpicture}
```



### 1.5.5 [shape=coordinate] ([1]17.2.1)

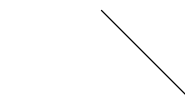
ノードに名前だけ付けて文字は書きたくない場合、困る場合がある。例えば次の例を考える。

```
\begin{tikzpicture}
\node (a0) at (0,0) {};
\node (a1) at (2,0) {};
\node (a2) at (1,1) {};
\draw (a0) --(a1) --(a2);
\end{tikzpicture}
```



3 点に名前を付け、折れ線を描いたつもりが、ノードに「大きさ」があるため線が途切れてしまう。このような場合に使えるのが [shape=coordinate] である。実際にこれを使うと次のようになる (shape= は省略可能であったことに注意)。

```
\begin{tikzpicture}
\node[coordinate] (a0) at (0,0) {};
\node[coordinate] (a1) at (2,0) {};
\node[coordinate] (a2) at (1,1) {};
\draw (a0) --(a1) --(a2);
\end{tikzpicture}
```



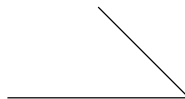
なお \node[coordinate] (名前) at (座標) {}; と書く代わりに

```
\path coordinate (名前) at (座標);
```

と書く事もできる。更に \path coordinate の略記として \coordinate を使う事がで

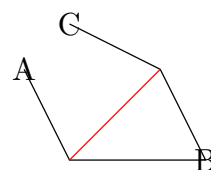
きる。よって上記の例は次のように書くことができる。

```
\begin{tikzpicture}
\coordinate (a0) at (0,0);
\coordinate (a1) at (2,0);
\coordinate (a2) at (1,1);
\draw (a0) --(a1) --(a2);
\end{tikzpicture}
```



この `coordinate` はオペレーションなので `node` オペレーションと同じように `\path` の途中で使うこともできる。

```
\begin{tikzpicture}
\draw (0,1) node{A} --(0.5,0) coordinate (X)
      --(2,0) node{B} --(1.5,1) coordinate (Y)
      --(0.5,1.5) node{C};
\draw[red] (X) --(Y);
\end{tikzpicture}
```



## 1.5.6 ノードのオプション

この節では `node` に付けられるオプションでまだ説明していないものについて説明する。

- `[inner sep=長さ]` ([1]17.2.3)

ノードの内側の余白の大きさを設定する (CSS での `padding` である)。初期値は `0.3333em` である。

```
\begin{tikzpicture}
\node[draw,inner sep=15pt] at (0,0) {AB};
\node[draw,inner sep=0pt] at (1.5,0) {XY};
\end{tikzpicture}
```



- `[inner xsep=長さ]` ([1]17.2.3)

ノードの内側の余白の大きさのうち、 $x$  軸方向のみを設定する。

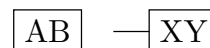
- `[inner ysep=長さ]` ([1]17.2.3)

ノードの内側の余白の大きさのうち、 $y$  軸方向のみを設定する。

- `[outer sep=長さ]` ([1]17.2.3)

ノードの外側の余白の大きさを設定する (CSS での `margin` である)。

```
\begin{tikzpicture}
\node[draw,outer sep=10pt] (a) at (0,0) {AB};
\node[draw,outer sep=0pt] (b) at (2,0) {XY};
\draw (a) --(b);
\end{tikzpicture}
```



- [outer xsep=長さ] ([1]17.2.3)  
ノードの外側の余白の大きさのうち、 $x$  軸方向のみを設定する。
- [outer ysep=長さ] ([1]17.2.3)  
ノードの外側の余白の大きさのうち、 $y$  軸方向のみを設定する。
- [minimum height=長さ] ([1]17.2.3)  
(inner sep を含めた) ノードの高さの最小値を設定する。初期値は 0pt である。
- [minimum width=長さ] ([1]17.2.3)  
(inner sep を含めた) ノードの幅の最小値を設定する。初期値は 0pt である。
- [minimum size=長さ] ([1]17.2.3)  
[minimum height] と [minimum width] に同じ値を設定する場合に使用できる。
- [behind path] ([1]17.2.1)  
ノードをパスの下に描くようにする。

```
\tikz \draw[draw=red!20,line width=10pt] (0,0)
      node[behind path]{text0} --(1,0) node{text1};
```

tex text1

- [in front of path] ([1]17.2.1)  
ノードをパスの上に描くようにする。通常はこの状態である。
- [text=色] ([1]17.4.1)  
文字の色を指定する。
- [font=フォントコマンド] ([1]17.4.2)  
ノードのフォントをコマンドで指定する。例えば次のような指定をすることができる。

```
\begin{tikzpicture}
\begin{tikzpicture}
\node[font=\tiny] at (0,1) {text2};
\node[font=\footnotesize] at (0,0.5) {text1};
\node[font=\itshape] at (0,0) {text0};
\end{tikzpicture}
```

text2

text1

text0

- [anchor=位置名] ([1]17.5.1)  
ノードのうちどの部分が（座標）にくるかを指定する。例えば [anchor=north] とすると、ノードの上の部分の点が、ノードの座標の位置になる。位置名 については 1.5.4 を参照。

```
\tikz \draw (0,0) node[anchor=north]{text0}
      --(1,1) node[anchor=south]{text1};
```

text1

text0

```
\tikz \draw (0,0) node{\$X\$} --(1.3,0)
      node{\$\displaystyle\prod_i Y_i\$};
```

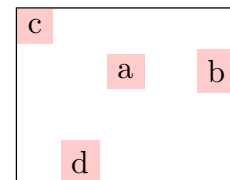
$$X \longrightarrow \prod_i Y_i$$

```
\tikz \draw (0,0) node[anchor=base]{\$X\$} --(1.3,0)
      node[anchor=base]{\displaystyle\prod_i Y_i\$};
```

$$X \longrightarrow \prod_i Y_i$$

- [above=長さ] ([1]17.5.2)  
既定値は 0pt であり，この場合 [anchor=south] と同じである．長さを指定した場合は，更にその分だけ上に移動する．
- [below=長さ] ([1]17.5.2)  
[above=長さ] と同様．
- [left=長さ] ([1]17.5.2)  
[above=長さ] と同様．
- [right=長さ] ([1]17.5.2)  
[above=長さ] と同様．
- [above left] ([1]17.5.2)  
[anchor=south east] と同じ．
- [above right] ([1]17.5.2)  
[above left] と同様．
- [below left] ([1]17.5.2)  
[above left] と同様．
- [below right] ([1]17.5.2)  
[above left] と同様．
- [centered] ([1]17.5.2)  
[anchor=center] と同じ．
- [fit=複数座標] ([1]17.6, 54) (\usetikzlibrary{fit} が必要)  
ノードの形が，指定した座標を全て含むように作られる．

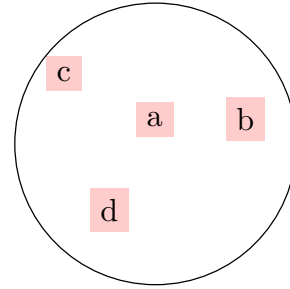
```
\begin{tikzpicture}
\node[fill=red!20] (a) at (0,0) {a};
\node[fill=red!20] (b) at (1,0) {b};
\node[fill=red!20] (c) at (-1,0.5) {c};
\node[fill=red!20] (d) at (-0.5,-1) {d};
\node[draw,inner sep=0pt,fit=(a)(b)(c)(d)] {};
\end{tikzpicture}
```



```

\begin{tikzpicture}
\node[fill=red!20] (a) at (0,0) {a};
\node[fill=red!20] (b) at (1,0) {b};
\node[fill=red!20] (c) at (-1,0.5) {c};
\node[fill=red!20] (d) at (-0.5,-1) {d};
\node[draw,shape=circle,inner sep=0pt,
      fit=(a)(b)(c)(d)] {};
\end{tikzpicture}

```



- [at start] ([1]17.8)  
[pos=0] と同じ.
- [very near start] ([1]17.8)  
[pos=0.125] と同じ.
- [near start] ([1]17.8)  
[pos=0.25] と同じ.
- [midway] ([1]17.8)  
[pos=0.5] と同じ.
- [near end] ([1]17.8)  
[pos=0.75] と同じ.
- [very near end] ([1]17.8)  
[pos=0.825] と同じ.
- [at end] ([1]17.8)  
[pos=1] と同じ.
- [auto=値] ([1]17.8)  
このオプションはノードが直線や曲線上にある場合のみ有効である. 値 には left, right, false のいずれかを設定する. [auto=left] とした場合, ノードが線の進行方向の左側に配置される. [auto=right] の場合は右になる. [auto=false] の場合はこの制御が無効になる. 通常は left が既定値である.

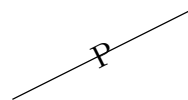
```
\tikz \draw (0,0) -- node[auto]{圏論} (2,1);
```

圏論

- [swap] ([1]17.8)  
auto の設定を left と right で入れ替える (使い方は第 1.8.2 節を参照). なお swap の略記として ' を使う事ができる.
- [sloped] ([1]17.8)

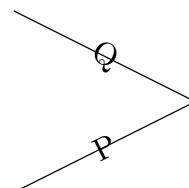
文字をパスの傾きに合わせて回転させる.

```
\tikz \draw (0,0) -- node[sloped]{P} (2,1);
```



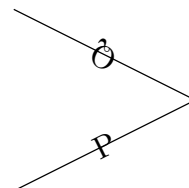
- [allow upside down=真偽値] ([1]17.8)  
[sloped] では文字は逆さまにならないようになっている.

```
\tikz \draw (0,0)
  -- node[sloped]{P} (2,1)
  -- node[sloped]{Q} (0,2);
```



[allow upside down=true] とすると文字が逆さまになるようになる. 初期値は false で既定値は true である.

```
\tikz \draw (0,0)
  -- node[sloped,allow upside down]{P} (2,1)
  -- node[sloped,allow upside down]{Q} (0,2);
```



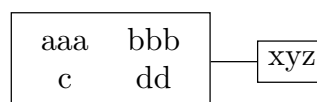
- [text opacity=割合] ([1]23.2)  
文字の不透明度については, 通常は [fill opacity] と同じ値がそのまま適用される. [fill opacity] とは別に文字の不透明度を設定したい場合に使用するのが [text opacity=割合] である.

## 1.5.7 ノードに複数行の文字を書く ([1]17.4.3)

ノードに複数行の文字を書きたい場合は以下のいずれかの方法を使う.

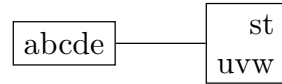
- (1) 複数行表示に使用する環境 (例えば tabular 環境) を置く.

```
\begin{tikzpicture}
\node[draw] (a) at (0,0) {
  \begin{tabular}{cc}
aaa & bbb\\
c   & dd
\end{tabular}
};
\node[draw] (b) at (2,0) {xyz};
\draw (a) --(b);
\end{tikzpicture}
```



- (2) `[align=値]` を指定した上で `\` を使い改行する (`[align]` を指定しないと改行されない).

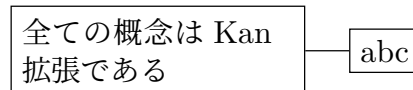
```
\begin{tikzpicture}
\node[draw] (a) at (0,0) {abc\\de};
\node[draw,align=right]
      (b) at (2,0) {st\\uvw};
\draw (a) --(b);
\end{tikzpicture}
```



値には `left`, `right`, `center` が設定できる. また `[align=none]` とするとこの制御が無効になる.

- (3) `[text width=長さ]` を指定する. 改行は幅に合わせて自動で行われる.

```
\begin{tikzpicture}
\node[draw,text width=3cm]
      (a) at (0,0)
      {全ての概念は Kan 拡張である};
\node[draw] (b) at (2.5,0) {abc};
\draw (a) --(b);
\end{tikzpicture}
```



またこれは `[align]` と組み合わせることもできる. 詳しくは [1]17.4.3 を参照.

## 1.6 Arrow Tips

`arrows` オプションを指定することでパスを「矢印」にすることができる. このオプションは `[arrows=開始部分-終了部分]` の形式で指定する (開始部分 と 終了部分 は空文字列にすることも可能). 例えば `\draw[arrows=|->] (0,0) --(2,0);` とすると次のようになる.



但し, `arrows` オプションについては `arrows=` を省略することが可能である<sup>\*12</sup>. 例えば先の例では `\draw[|->] (0,0) --(2,0);` としてもよい.

開始部分 と 終了部分 には次のようなものが使用できる.

```
—————| \draw[|-|] (0,0) --(1,0);
<—————> \draw[<->] (0,0) --(1,0);
>—————< \draw[>-<] (0,0) --(1,0);
```

<sup>\*12</sup> - を含むオプションは `arrows` オプションと解釈される



更に `\usetikzlibrary{arrows.meta}` とする<sup>\*13</sup>ことで、以下のようなものも使う事ができる (下記以外にも色々あるので詳しくは [1]16.5 を参照).

```

 $\longrightarrow$  \draw[Arc Barb-Arc Barb] (0,0) --(1,0);
 $\lrcorner$  \draw[Bracket-Bracket] (0,0) --(1,0);
 $\hookrightarrow$  \draw[Hooks-Hooks] (0,0) --(1,0);
 $\longleftarrow$  \draw[Straight Barb-Straight Barb] (0,0) --(1,0);
 $\lceil$  \draw[Tee Barb-Tee Barb] (0,0) --(1,0);
 $\longrightarrow$  \draw[Classical TikZ Rightarrow-Classical TikZ Rightarrow] (0,0) --(1,0);
 $\longrightarrow$  \draw[Computer Modern Rightarrow-Computer Modern Rightarrow] (0,0) --(1,0);
 $\bullet$  \draw[Circle-Circle] (0,0) --(1,0);
 $\blacklozenge$  \draw[Diamond-Diamond] (0,0) --(1,0);

```

開始部分 と 終了部分 には複数指定することもできる.

```

 $\longrightarrow$  \draw[->>] (0,0) --(1,0);
 $\longrightarrow$  \draw[->>>] (0,0) --(1,0);
 $\longrightarrow$  \draw[->>>>] (0,0) --(1,0);

```

この場合、を使う事で、パスを途中で止めることができる.

```

 $\longrightarrow$  \draw[->.>>] (0,0) --(1,0);
 $\longrightarrow$  \draw[->>.>] (0,0) --(1,0);

```

また 開始部分 と 終了部分 にはオプションを設定することもできる (詳しくは [1]16 を参照).

```

 $\longrightarrow$  \draw[{Hooks[right]}-{Stealth[color=red]}] (0,0) --(1,0);

```

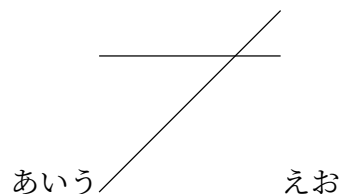
## 1.7 baseline オプション ([1]12.2.1)

最初に説明した通り、`tikzpicture` 環境を本文中に書くと、通常は `tikzpicture` 環境の一番下の部分が本文の baseline の位置に来る. これを変更するオプションが `[baseline=(座標)]` で、(座標) で設定した位置が baseline の位置に来る.

```

あいう
\begin{tikzpicture}
\draw (0,-1) --(2,1);
\draw (0,0.5) --(2,0.5);
\end{tikzpicture}%
えお

```

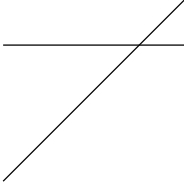


<sup>\*13</sup> `arrows` や `arrows.spaced` というライブラリもあるが、これらは古く非推奨となっている. また小文字で始まる 開始部分 や 終了部分 も古いものであり非推奨である. [1]16.1 の Remark を参照.

```

あいう
\begin{tikzpicture}[baseline={(0,0.5)}] あいう _____ えお
\draw (0,-1) --(2,1);
\draw (0,0.5) --(2,0.5);
\end{tikzpicture}%
えお

```




(座標) の設定に位置名 `base` を使う (第 1.5.4 節を参照) と、これはノードの文字の `baseline` を指すので、文字の位置を簡単に揃えることができる。

```

あい
\begin{tikzpicture}[baseline=(a.base)]
\node[draw,inner sep=7pt] (a) at (0,0) {う}; あい う えお
\end{tikzpicture}%
えお

```

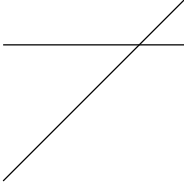


このオプションは `[baseline=長さ]` という形式での設定もすることができる。この設定は `[baseline={(0pt, 長さ)}]` と同様の動きをする。

```

あいう
\begin{tikzpicture}[baseline=5mm]
\draw (0,-1) --(2,1);
\draw (0,0.5) --(2,0.5);
\end{tikzpicture}%
えお

```



また `[baseline=default]` と設定すると元々の動きになる。

## 1.8 オプション指定時に使える便利な機能

例えば赤く太い矢印を複数描く場合、次のように同じオプション指定を何度も書くことになってしまう。

```

\begin{tikzpicture}
\draw[->,color=red,thick] (0,0) --(15:1);
\draw[->,color=red,thick] (0,0) --(75:1);
\draw[->,color=red,thick] (0,0) --(135:1);
\end{tikzpicture}

```



このような場合に使える便利な方法がいくつかあるので、ここではそれを説明する。

### 1.8.1 スタイル ([1]12.4, 87.4.4)

「スタイル」とは、いくつかのオプション設定にまとめて名前をつけるようなものである。tikzpicture 環境のオプションに [スタイル名/.style=設定内容] と書くと、設定内容にスタイル名という名前を付けることができる。こうするとオプションに [スタイル名] と書くだけで設定内容が反映される。例えば次の例では、->,color=red,thick に red arrow という名前を付けているので、\draw のオプションに単に [red arrow] と書くだけで赤く太い矢印にすることができる。

```
\begin{tikzpicture}[red arrow/.style={->,color=red,thick}]
\draw[red arrow] (0,0) --(15:1);
\draw[red arrow] (0,0) --(75:1);
\draw[red arrow] (0,0) --(135:1);
\end{tikzpicture}
```



但しこの書き方では、[red arrow] を使えるのはこの tikzpicture 環境だけである。他の tikzpicture 環境でも使用できるようにしたい場合は \tikzset というコマンドを使用する。この中にスタイルの定義を書くと、全ての tikzpicture 環境でそのスタイルが使用できるようになる (通常はプリアンブルなどに予め \tikzset を書いておくことになるであろう)。

```
\tikzset{red arrow/.style={->,color=red,thick}}

\begin{tikzpicture}
\draw[red arrow] (0,0) --(15:1);
\draw[red arrow] (0,0) --(75:1);
\draw[red arrow] (0,0) --(135:1);
\end{tikzpicture}
```



スタイルでは引数を 1 つ使うことができる。その場合は通常の T<sub>E</sub>X のように #1 を使う。引数に値を入れるには、オプションに指定する際に [スタイル名=値] のように書く。

```
\begin{tikzpicture}[outline/.style={draw=#1,fill=#1!20}]
\node[outline=red] at (0,1) {red box};
\node[outline=blue] at (0,0) {blue box};
\end{tikzpicture}
```

red box

blue box

更にこの場合 [スタイル名/.default=既定値] により既定値を設定することもできる。

```
\begin{tikzpicture}[outline/.style={draw=#1,fill=#1!20},
outline/.default=red]
\node[outline] at (0,1) {red box};
\node[outline=blue] at (0,0) {blue box};
\end{tikzpicture}
```

red box

blue box

引数を 2 つ以上にすることも可能である ([1]87.4.4 を参照)。

`/.style` による設定は上書きされる．設定を追加したい場合は `/.append style` を使う ([1]87.4.4).

```
\tikzset{arrowStyle/.style={->,color=red}}
\tikz \draw[arrowStyle] (0,0) --(0,1);
\tikzset{arrowStyle/.style={->,color=blue}}
\tikz \draw[arrowStyle] (0,0) --(0,1);
\tikzset{arrowStyle/.append style=thick}
\tikz \draw[arrowStyle] (0,0) --(0,1);
```

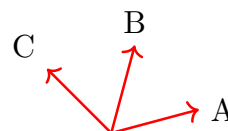


`/.append style` は設定が後ろに追加される．前に追加するための `/.prefix style` というものも用意されている ([1]87.4.4).

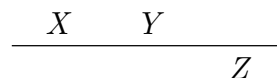
## 1.8.2 scope 環境 ([1]12.3)

TikZ では「外側」に指定されたオプションは「内側」のそれぞれの要素に適用されるようになっている．そこで次のように、`\path` のオプションを `tikzpicture` 環境のオプションとして設定したり、`node` のオプションを `\path` や `tikzpicture` 環境に設定することができる．

```
\begin{tikzpicture}[->,draw=red,thick]
\draw (0,0) --( 15:1) node[right]{A};
\draw (0,0) --( 75:1) node[above]{B};
\draw (0,0) --(135:1) node[above left]{C};
\end{tikzpicture}
```

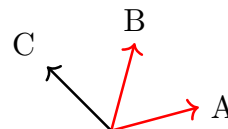


```
\tikz \draw[auto,font=\itshape] (0,0)
-- node{X} (1,0) -- node{Y} (2,0)
-- node[swap]{Z} (3,0);
```



但し `tikzpicture` 環境にオプションを設定してしまうと、全体に同じオプションが適用されてしまう．そこで使えるのが `scope` 環境である．

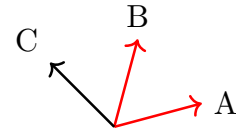
```
\begin{tikzpicture}[->,thick]
\begin{scope}[draw=red]
\draw (0,0) --( 15:1) node[right]{A};
\draw (0,0) --( 75:1) node[above]{B};
\end{scope}
\draw (0,0) --(135:1) node[above left]{C};
\end{tikzpicture}
```



更に `\usetikzlibrary{scopes}` を使うと `\begin{scope}`, `\end{scope}` を単に `{, }`

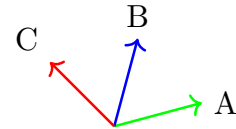
と書けるようになる.

```
\begin{tikzpicture}[->,thick]
{[draw=red]
\draw (0,0) --( 15:1) node[right]{A};
\draw (0,0) --( 75:1) node[above]{B};
}
\draw (0,0) --(135:1) node[above left]{C};
\end{tikzpicture}
```



なおこの設定方法では、より「内側」の設定の方が優先される.

```
\begin{tikzpicture}[->,draw=red,thick]
\begin{scope}[draw=blue]
\draw[draw=green] (0,0) --(15:1) node[right]{A};
\draw (0,0) --( 75:1) node[above]{B};
\end{scope}
\draw (0,0) --(135:1) node[above left]{C};
\end{tikzpicture}
```



また `scope` 環境に設定するオプションもある.

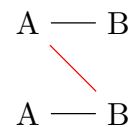
- `[name prefix=名前]` ([1]17.2.1)

次の例を参照.

```
\begin{tikzpicture}
\begin{scope}[name prefix=top-]
\node (A) at (0,1) {A};
\node (B) at (1,1) {B};
\draw (A) --(B);
\end{scope}

\begin{scope}[name prefix=bottom-]
\node (A) at (0,0) {A};
\node (B) at (1,0) {B};
\draw (A) --(B);
\end{scope}

\draw[red] (top-A) --(bottom-B);
\end{tikzpicture}
```



- `[name suffix=名前]` ([1]17.2.1)  
`[name prefix=名前]` と同様.
- `[transparency group]` ([1]23.5)

設定した範囲を1つのまとまりとして、opacity を適用する。

```
\begin{tikzpicture}[opacity=0.5]
\draw[line width=5mm] (0,0) --(1,1);
\draw[line width=5mm] (1,0) --(0,1);
\end{tikzpicture}
```



```
\begin{tikzpicture}[opacity=0.5,transparency group]
\draw[line width=5mm] (0,0) --(1,1);
\draw[line width=5mm] (1,0) --(0,1);
\end{tikzpicture}
```

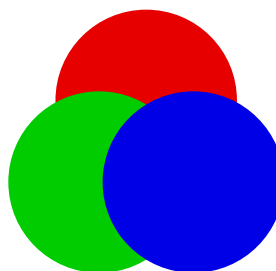


より詳しい設定もできるが、詳しくは [1]23.5 を参照。

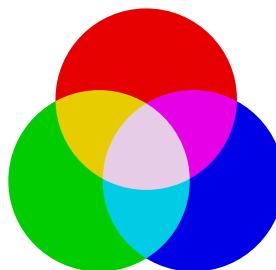
- [blend group=モード] ([1]23.3)

色を重ねた場合、通常は後から描いた方が優先されるが、色を混ぜることもできる。それが [blend group=モード] で、モード によりどのように混ぜるかを設定する。詳しくは [1]23.3 を参照。

```
\begin{tikzpicture}[radius=1cm]
\fill[red!90!black] ( 90:0.6) circle;
\fill[green!80!black] (210:0.6) circle;
\fill[blue!90!black] (330:0.6) circle;
\end{tikzpicture}
```



```
\begin{tikzpicture}[radius=1cm,
blend group=screen]
\fill[red!90!black] ( 90:0.6) circle;
\fill[green!80!black] (210:0.6) circle;
\fill[blue!90!black] (330:0.6) circle;
\end{tikzpicture}
```



なおいくつかのオプションについては、外側のオプションをそのまま内側に適用するのではない場合がある。

- [draw], [fill], [shade]

第 1.4 節でも説明したように、[draw] は外側に設定した場合は既定値を変更する効果がある。これは [fill] や shade 関連のオプションについても同様である。

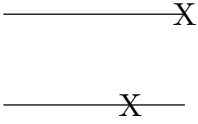
```
\begin{tikzpicture}[top color=red]
\draw (0,0) rectangle(1,1);
\shade (2,0) rectangle(3,1);
\end{tikzpicture}
```



- [pos] ([1]17.9)

外側に指定した場合は node を中間に置いた場合にのみ適用される.

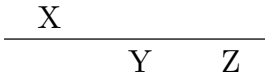
```
\begin{tikzpicture}
\draw[pos=0.7] (0,1) --(2,1) node{X};
\draw[pos=0.7] (0,0) -- node{X} (2,0);
\end{tikzpicture}
```



- [auto] ([1]17.8)

内側にも [auto] を適用した上で、既定値も変更する.

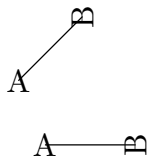
```
\tikz \draw[auto=right] (0,0)
-- node[auto=left]{X} (1,0)
-- node[auto]{Y} (2,0)
-- node{Z} (3,0);
```



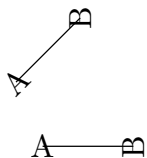
- [transform shape] ([1]17.7)

ノードに対する [scale] や [rotate] などの操作は、外側に書いた場合は通常は適用されない。適用されるようにしたい場合はそのノードに [transform shape] を書く.

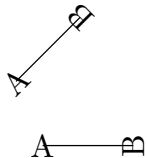
```
\begin{tikzpicture}
\draw (1,0) node{A} --(2,0) node[rotate=90]{B};
\draw[rotate=45]
(1,0) node{A} --(2,0) node[rotate=90]{B};
\end{tikzpicture}
```



```
\begin{tikzpicture}
\draw (1,0) node{A} --(2,0) node[rotate=90]{B};
\draw[rotate=45]
(1,0) node[transform shape]{A}
--(2,0) node[rotate=90]{B};
\end{tikzpicture}
```




```
\begin{tikzpicture}
\draw (1,0) node{A} --(2,0) node[rotate=90]{B};
\draw[rotate=45,transform shape]
(1,0) node{A} --(2,0) node[rotate=90]{B};
\end{tikzpicture}
```



- [preaction], [postaction] ([1]15.10)

このオプションは直接 \path に設定する必要がある、tikzpicture 環境や scope 環境に設定しても適用されない.

```
\begin{tikzpicture}[preaction={draw=red,line width=5pt}]
\draw (0,0) --(1,0);
\end{tikzpicture}
```

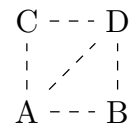


### 1.8.3 every

TikZ には特別なスタイル名として `every path` というものが用意されている。このスタイルは全てのパスに適用される ([1]14)。これにより `[every path/.style=設定]` のように書くことで全てのパスに `設定` を適用させることができる。例えば次のように書くと全てのパスが点線になる。

```
\begin{tikzpicture}[every path/.style=dashed]
\node (A) at (0,0) {A};
\node (B) at (1,0) {B};
\node (C) at (0,1) {C};
\node (D) at (1,1) {D};

\draw (A) --(B);
\draw (A) --(C);
\draw (A) --(D);
\draw (B) --(D);
\draw (C) --(D);
\end{tikzpicture}
```

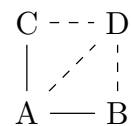


第 1.8.2 節で述べた `scope` 環境と組み合わせれば、特定の範囲だけ `設定` を適用することもできる。

```
\begin{tikzpicture}
\node (A) at (0,0) {A};
\node (B) at (1,0) {B};
\node (C) at (0,1) {C};
\node (D) at (1,1) {D};

\draw (A) --(B);
\draw (A) --(C);

\begin{scope}[every path/.style=dashed]
\draw (A) --(D);
\draw (B) --(D);
\draw (C) --(D);
\end{scope}
\end{tikzpicture}
```



また、例えば全ての `tikzpicture` 環境の全てのパスに `[line width=1pt]` を適用したい場合は

```
\tikzset{every path/.style={line width=1pt}}
```

と書くことができる。また `/.style` だけでなく `/.append style` なども使用できる。



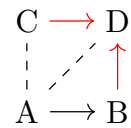
次の例を参照.

```
\begin{tikzpicture}[every path/.style=->]
\node (A) at (0,0) {A};
\node (B) at (1,0) {B};
\node (C) at (0,1) {C};
\node (D) at (1,1) {D};

\draw (A) --(B);

\begin{scope}[every path/.style=dashed]
\draw (A) --(C);
\draw (A) --(D);
\end{scope}

\begin{scope}[every path/.append style=red]
\draw (B) --(D);
\draw (C) --(D);
\end{scope}
\end{tikzpicture}
```



このような every ○○ というものは他にも用意されている.

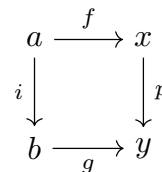
- every picture ([1]12.2.1)  
全ての tikzpicture 環境に適用されるスタイルを表す. これは主に \tikzset で使用する.
- every circle ([1]14.6)  
全ての circle に適用されるスタイルを表す.
- every to ([1]14.13)  
全ての to に適用されるスタイルを表す.
- every node ([1]17.2.1)  
全ての node に適用されるスタイルを表す.
- every rectangle node ([1]17.2.1)  
[shape=rectangle] となっている全ての node に適用されるスタイルを表す.
- every circle node ([1]17.2.1)  
[shape=circle] となっている全ての node に適用されるスタイルを表す.
- every label ([1]17.10.2)  
label オプションで作られた全てのノードに適用されるスタイルを表す.
- every pin ([1]17.10.3)  
pin オプションで作られた全てのノードに適用されるスタイルを表す.

- `every pin edge` ([1]17.10.3)  
pin オプションで作られた全ての edge に適用されるスタイルを表す.
- `every edge` ([1]17.12.1)  
全ての edge に適用されるスタイルを表す.
- `every scope` ([1]12.3.1)  
全ての scope 環境に適用されるスタイルを表す.

## 1.9 圏論で図式を描く例

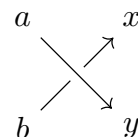
次は普通の四角い可換図式の例である.

```
\begin{tikzpicture}[auto,x=12mm,y=12mm,->]
\node (a) at (0,1) {$a$}; \node (x) at (1,1) {$x$};
\node (b) at (0,0) {$b$}; \node (y) at (1,0) {$y$};
\draw (a) -- node[scriptstyle f] (x);
\draw (x) -- node[scriptstyle p] (y);
\draw (a) -- node[swap] [scriptstyle i] (b);
\draw (b) -- node[swap] [scriptstyle g] (y);
\end{tikzpicture}
```



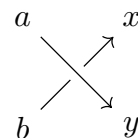
線を立体的に交差させたい場合は, 次のように白い太線を使えばよい.

```
\begin{tikzpicture}[x=12mm,y=12mm]
\node (a) at (0,1) {$a$}; \node (x) at (1,1) {$x$};
\node (b) at (0,0) {$b$}; \node (y) at (1,0) {$y$};
\draw[->] (b) --(x);
\draw[white,line width=6pt] (a) --(y);
\draw[->] (a) --(y);
\end{tikzpicture}
```



これは `[preaction]` を使うと次のように書ける.

```
\tikzset{cross/.style={preaction={-,draw=white,line width=6pt}}}
\begin{tikzpicture}[x=12mm,y=12mm]
\node (a) at (0,1) {$a$}; \node (x) at (1,1) {$x$};
\node (b) at (0,0) {$b$}; \node (y) at (1,0) {$y$};
\draw[->] (b) --(x);
\draw[->,cross] (a) --(y);
\end{tikzpicture}
```



`equalizer` の時に使うような, 平行射を描きたい場合は `[transform canvas]` を使うと

上手くいく.

```
\begin{tikzpicture}[auto,x=12mm,->]
\node (a) at (0,0) {$a$}; \node (x) at (1,0) {$x$};
\draw[transform canvas={yshift=2pt}]
(a) -- node{$\scriptstyle f$} (x);
\draw[transform canvas={yshift=-2pt}]
(a) -- node[swap]{$\scriptstyle g$} (x);
\end{tikzpicture}
```

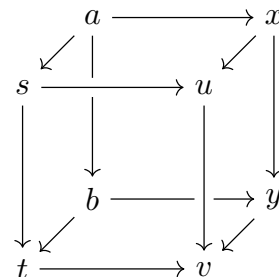
$$a \begin{array}{c} \xrightarrow{f} \\ \xleftarrow{g} \end{array} x$$

立体的な可換図式を描く場合, xyz 座標系を使う事もできる.

```
\tikzset{cross/.style={preaction={-,draw=white,line width=6pt}}}
\begin{tikzpicture}[->]
\node (a) at (0,2,0) {$a$}; \node (x) at (2,2,0) {$x$};
\node (b) at (0,0,0) {$b$}; \node (y) at (2,0,0) {$y$};
\node (s) at (0,2,2) {$s$}; \node (u) at (2,2,2) {$u$};
\node (t) at (0,0,2) {$t$}; \node (v) at (2,0,2) {$v$};
\draw (a) --(x);
\draw (x) --(y);
\draw (a) --(b);
\draw (b) --(y);

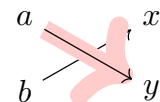
\draw[cross] (s) --(u);
\draw[cross] (u) --(v);
\draw (s) --(t);
\draw (t) --(v);

\draw (a) --(s);
\draw (b) --(t);
\draw (x) --(u);
\draw (y) --(v);
\end{tikzpicture}
```



ちなみに cross の定義に - を含めている理由は, これを入れておかないと元のパスの [->] の影響を受けてしまうからである. 次の例を参照.

```
\tikzset{cross2/.style={preaction={draw=red!20,line width=6pt}}}
\begin{tikzpicture}[x=14mm,y=8mm]
\node (a) at (0,1) {$a$}; \node (x) at (1,1) {$x$};
\node (b) at (0,0) {$b$}; \node (y) at (1,0) {$y$};
\draw[->] (b) --(x);
\draw[->,cross2] (a) --(y);
\end{tikzpicture}
```



## 第 2 章

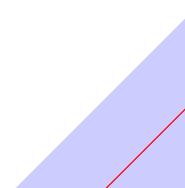
# より高度な内容

### 2.1 newcommand について

当たり前だが TikZ も TeX なので、`\newcommand` により新しいコマンドを定義することもできる。例えば次は指定した 3 点で三角形を描くコマンドである。

```
\newcommand{\pathTriangle}[4][]{\begin{tikzpicture}
  \path[#1] (#2) --(#3) --(#4) --cycle;
\end{tikzpicture}}
```

```
\begin{tikzpicture}
\pathTriangle[fill=blue!20]{0,0}{2,0}{2,2};
\pathTriangle[draw=red]{1,0}{2,0}{2,1};
\end{tikzpicture}
```



また次は文字に赤いバツ印を付けるコマンドの例である。

```
\newcommand{\batsu}[1]{\begin{tikzpicture}[baseline=(a.base)]
\node[inner sep=0pt] (a) at (0,0) {#1};
\draw[red] (a.south east) --(a.north west);
\draw[red] (a.south west) --(a.north east);
\end{tikzpicture}}
```

あいうえお\batsu{かきくけこ}さしすせそ

あいうえお~~かきくけこ~~さしすせそ

## 2.2 色の設定について

この節では [draw] や [fill] 等で設定できる色について説明する\*<sup>1</sup>.

### 2.2.1 色の指定の仕方について

まず良く使われる色については名前が定義されており、これはそのまま使うことができる. 使うことができる名前は以下の通り\*<sup>2</sup>([2]2.4.1).

|                                                                                   |         |                                                                                   |          |                                                                                   |           |                                                                                     |       |
|-----------------------------------------------------------------------------------|---------|-----------------------------------------------------------------------------------|----------|-----------------------------------------------------------------------------------|-----------|-------------------------------------------------------------------------------------|-------|
|  | red     |  | green    |  | blue      |  | cyan  |
|  | magenta |  | yellow   |  | black     |  | gray  |
|                                                                                   | white   |  | darkgray |  | lightgray |  | brown |
|  | lime    |  | olive    |  | orange    |  | pink  |
|  | purple  |  | teal     |  | violet    |                                                                                     |       |

また色名に - を付けることで補色を表すことができる.

```
\tikz \path[fill=-red] (0,0) rectangle(1,0.5);
```



更にこれらの色は混ぜて使うことができる. これは standard color expression とい

色名<sub>0</sub>!百分率<sub>1</sub>!色名<sub>1</sub>!百分率<sub>2</sub>!...!百分率<sub>n</sub>!色名<sub>n</sub>

という形式で書く ([2]2.3.2). これは 色名<sub>k</sub> には上記の名前 (- も使用可能) を設定して 百分率<sub>k</sub> には 0 ~ 100 を設定する. これは式で書くと

$$\sum_{i=0}^n \left( \prod_{j=i+1}^n \frac{\text{百分率}_j}{100} \right) \left( 1 - \frac{\text{百分率}_i}{100} \right) \text{色名}_i$$












という色を表す (但し 百分率<sub>0</sub> := 0 としておく)([2]6.1). 例えば red!70!green は red を 7 割, green を 3 割の割合で混ぜた色を表す (混ぜるの意味については第 2.2.2 節も参照).

```
\tikz \path[fill=red!70!green] (0,0) rectangle(1,0.5);
```



\*<sup>1</sup> これは xcolor パッケージで使えるものと同じであり詳しくはそのマニュアル [2] を参照 (コマンドプロンプトで「texdoc xcolor」を実行しても読める).

\*<sup>2</sup> これらは texmf-dist\tex\latex\color\xcolor\xcolor.sty で定義されている.

|                                                                                   |               |                                                                                   |               |                                                                                    |               |
|-----------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------------------------|---------------|------------------------------------------------------------------------------------|---------------|
|  | red!100!black |  | red!100!white |  | red!100!green |
|  | red!70!black  |  | red!70!white  |  | red!70!green  |
|  | red!30!black  |  | red!30!white  |  | red!30!green  |
|  | red!0!black   |                                                                                   | red!0!white   |  | red!0!green   |

また最後の 色名<sub>n</sub> は省略することも出来て、その場合は 色名<sub>n</sub> として white を指定した場合と同じ色になる.


|                                                                                   |         |                                                                                   |        |                                                                                   |        |                                                                                     |       |
|-----------------------------------------------------------------------------------|---------|-----------------------------------------------------------------------------------|--------|-----------------------------------------------------------------------------------|--------|-------------------------------------------------------------------------------------|-------|
|  | red!100 |  | red!70 |  | red!30 |  | red!0 |
|-----------------------------------------------------------------------------------|---------|-----------------------------------------------------------------------------------|--------|-----------------------------------------------------------------------------------|--------|-------------------------------------------------------------------------------------|-------|

色を混ぜる方法はもう 1 つあり、それを extended color expression という. これは

形式, 値: 色<sub>1</sub>, 値<sub>1</sub>; 色<sub>2</sub>, 値<sub>2</sub>; …; 色<sub>n</sub>, 値<sub>n</sub>


という形式で書き ([2]2.3.3), 色<sub>k</sub> を 値<sub>k</sub>/値 の割合で混ぜた色を表す. 形式 には rgb, cmy, cmyk, hsb, gray のいずれかを設定し, 色をどのように混ぜるかを表す (詳しくは 第 2.2.2 節も参照). 例えば色を RGB で指定したい場合は次のように書くことができる.

```
\tikz \path[fill={rgb,255:red,86;green,180;blue,233}]
(0,0) rectangle(1,0.5);
```




色<sub>k</sub> の部分には色名だけではなく, 上記の standard color expression も書くことができる.

```
\tikz \path[fill={rgb,3:red!30!green,2;-blue,1}]
(0,0) rectangle(1,0.5);
```




また 値 = 値<sub>1</sub> + … + 値<sub>n</sub> のときは, 値 を省略することができる.

```
\tikz \path[fill={rgb:red,4;green,2;yellow,1}]
(0,0) rectangle(1,0.5);
```




混ぜた色には \colorlet{色名}{色} で名前をつけて使用することもできる.

```
\colorlet{mycolor1}{red!30}
\tikz \path[fill=mycolor1] (0,0) rectangle(1,0.5);
```



```
\colorlet{mycolor2}{rgb:red,4;green,2;yellow,1}
\tikz \path[fill=mycolor2] (0,0) rectangle(1,0.5);
```



## 2.2.2 色の定義と形式について

色の名前は自分で直接定義することもできる。そのためのコマンドが `\definecolor` で `\definecolor{色名}{形式}{パラメーター}` の形式で使用する ([2]2.5.2)。色名 が定義する名前で、形式 には パラメーター をどのような形式で与えるかを設定する。形式 には次のものを使うことができる。

- **rgb**

色を RGB で指定する。パラメーター には赤、緑、青の量をそれぞれ 0 ～ 1 で、カンマ区切りで設定する。

```
\definecolor{darkgreen}{rgb}{0,0.39,0}
\tikz \path[fill=darkgreen] (0,0) rectangle(1,0.5);
```



- **RGB**

これも RGB だが、こちらの パラメーター には 0 ～ 255 の整数値で設定する。

```
\definecolor{aquamarine}{RGB}{127,255,212}
\tikz \path[fill=aquamarine] (0,0) rectangle(1,0.5);
```



- **HTML**

これも RGB だが、HTML で使用するような 000000 ～ FFFFFFFF で設定する。アルファベットは大文字でも小文字でも可。

```
\definecolor{gold}{HTML}{FFD700}
\tikz \path[fill=gold] (0,0) rectangle(1,0.5);
```



- **cmY**

色を CMY で指定する。パラメーター にはシアン、マゼンタ、黄の量をそれぞれ 0 ～ 1 で、カンマ区切りで設定する。

- **cmYk**

色を CMYK で指定する。パラメーター にはシアン、マゼンタ、黄、黒の量をそれぞれ 0 ～ 1 で、カンマ区切りで設定する。

- **hsb**

色を HSB で指定する。パラメーター には色相、彩度、明度をそれぞれ 0 ～ 1 で、カンマ区切りで設定する。

- **Hsb**

これも HSB だが、色相を 0 ～ 360 の角度で設定する (彩度、明度は 0 ～ 1)。

- **wave**

色を波長で指定する。パラメーター には波長を 363 ～ 814 (単位は nm) で設定する。

- `gray`  
灰色を表し、パラメーターには黒の量を 0 ～ 1 で設定する.
- `Gray`  
これも灰色を表し、パラメーターには黒の量を 0 ～ 15 の整数値で設定する.

また同様のコマンドとして `\providecolor` がある. これは `\definecolor` と同じだが, 色名 が定義されていない場合だけ実行される (`\definecolor` の場合は定義が上書きされる).

そして色を混ぜる際は, 1 つの形式に変換してから混ぜる処理を行う (変換の計算式は [2]6.3 を参照). `extended color expression` の場合は 形式 で指定した形式を使用する. `standard color expression` では 色名<sub>0</sub> の形式で処理を行う. なお最初に述べた色名については, それぞれ次のようになっている.

- `rgb` で定義されている: `red`, `green`, `blue`, `brown`, `lime`, `orange`, `pink`, `purple`, `teal`, `violet`
- `cmYk` で定義されている: `cyan`, `magenta`, `yellow`, `olive`
- `gray` で定義されている: `black`, `darkgray`, `gray`, `lightgray`, `white`

従って `green!50!cyan` の場合は `rgb` で色を混ぜるが, `cyan!50!green` の場合は `cyan` で色を混ぜることになる.

```
\tikz \path[fill=green!50!cyan] (0,0) rectangle(1,0.5);
```



```
\tikz \path[fill=cyan!50!green] (0,0) rectangle(1,0.5);
```



特定の形式に色を変換したい場合には第 2.2.1 節で述べた `\colorlet` を使うことができる. `\colorlet` はオプション引数で使用する形式を指定できる.

```
\colorlet{rgbcyan}{rgb}{cyan}
\begin{tikzpicture}
\path[fill=cyan] (0,3) rectangle(1,3.5);
\path[fill=rgbcyan] (0,2) rectangle(1,2.5);
\path[fill=green!50!rgbcyan] (0,1) rectangle(1,1.5);
\path[fill=rgbcyan!50!green] (0,0) rectangle(1,0.5);
\end{tikzpicture}
```





色の形式を gray にすることでグレースケールを得るという使い方もできる.

```
\colorlet{graycyan}[gray]{cyan}
\begin{tikzpicture}
\tikz \path[fill=graycyan] (0,0) rectangle(1,0.5);
\end{tikzpicture}
```

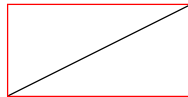


## 2.3 bounding box ([1]15.8)

tikzpicture 環境が作る領域を bounding box という. 例えば

```
\tikz \draw (0,0) --(2,1);
```

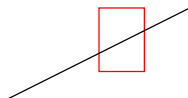
の場合, bounding box は次の赤枠で示した四角となる.



tikzpicture 環境はこの bounding box を元にして文章中に配置される. bounding box は, 通常はパス全体がぴったり収まるよう自動的に計算されるが, 明示的に指定することもできる. そのためのオプションが [use as bounding box] であり, このオプションを付けた \path より後の \path は bounding box の計算時に無視される. 例えば

```
本文中にはこのように
\begin{tikzpicture}
\path[use as bounding box] (1,0.3) rectangle(1.5,1);
\draw (0,0) --(2,1);
\end{tikzpicture}%
表示される.
```

のように書くと, この場合の bounding box は



の赤枠部分になるから, 本文中にはこのように 表示される. なお

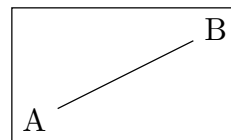
```
\path[use as bounding box]
```

の略記として \useasboundingbox が用意されている.

また (current bounding box) で現在の bounding box を表すノードを指定することができる ([1]106.4). このノードは [shape=rectangle] である (よってその anchor

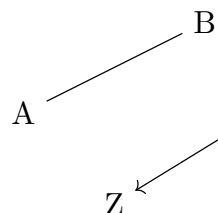
も使用できる). 例えば次のように書くことで全体を囲う枠を描くことができる.

```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (2,1) {B};
\draw (a) --(b);
\draw (current bounding box.south west)
      rectangle(current bounding box.north east);
\end{tikzpicture}
```



また `scope` 環境に対しては `[local bounding box=名前]` とすることで, その `scope` 環境が作る bounding box に名前を付けて参照することができる ([1]106.4).

```
\begin{tikzpicture}
\begin{scope}[local bounding box=test]
\node (a) at (0,1) {A};
\node (b) at (2,2) {B};
\end{scope}
\node (z) at (1,0) {Z};
\draw (a) --(b);
\draw[->] (test.south east) --(z);
\end{tikzpicture}
```

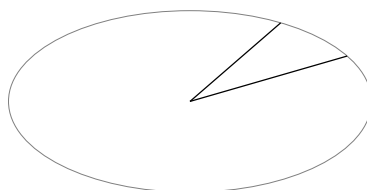


## 2.4 座標について

### 2.4.1 極座標系について ([1]13.2.1)

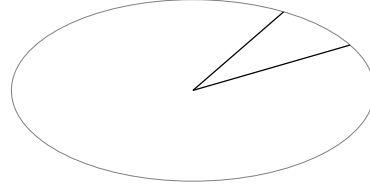
極座標を表す座標系名 `canvas polar` では, 座標として円だけではなく楕円も使用することができる. この場合 `radius` ではなく `x radius` と `y radius` を設定する.

```
\begin{tikzpicture}
\draw[help lines] (0,0) circle[x radius=2cm,y radius=1cm];
\draw (0,0) --(canvas polar cs:angle=30,x radius=2cm,y radius=1cm);
\draw (0,0) --(canvas polar cs:angle=60,x radius=2cm,y radius=1cm);
\end{tikzpicture}
```



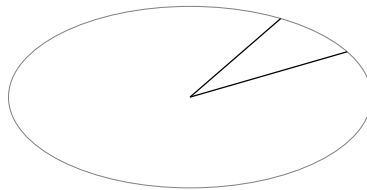
略記する場合は (角度: 長さ<sub>x</sub> and 長さ<sub>y</sub>) と書く.

```
\begin{tikzpicture}
\draw[help lines] (0,0)
  circle[x radius=2cm,y radius=1cm];
\draw (0,0) --(30:2cm and 1cm);
\draw (0,0) --(60:2cm and 1cm);
\end{tikzpicture}
```



また似たような座標系名として `xyz polar` がある. これも極座標系だが, `x radius` と `y radius` の大きさの単位として `[x]` と `[y]` を使用する.

```
\begin{tikzpicture}[x=2cm,y=1cm]
\draw[help lines] (0,0) circle[x radius=2cm,y radius=1cm];
\draw (0,0) --(xyz polar cs:angle=30,radius=1);
\draw (0,0) --(xyz polar cs:angle=60,radius=1);
\end{tikzpicture}
```



これも `canvas polar` と同様の略記をすることができる. これは長さの単位がなければ `xyz polar`, 単位があれば `canvas polar` として扱われる.

```
\begin{tikzpicture}[x={(0cm,1cm)},y={(-1cm,0cm)}]
\draw[->,red] (0,0) --(1,0);
\draw (0,0) --(30:1cm) --(60:1cm) --(90:1cm);
\end{tikzpicture}
```



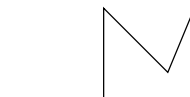
```
\begin{tikzpicture}[x={(0cm,1cm)},y={(-1cm,0cm)}]
\draw[->,red] (0,0) --(1,0);
\draw (0,0) --(30:1) --(60:1) --(90:1);
\end{tikzpicture}
```



## 2.4.2 相対的な座標指定 ([1]13.4.1)

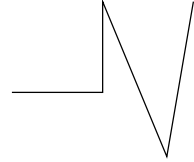
第 1.2 節で「座標」の指定方法について述べたが, これらは絶対的な座標指定である. (座標) の前に `++` を付けて `++(座標)` とすると, これは「直前の座標」からの相対的な座標指定になる.

```
\tikz \draw (0,0) --(1,0) ---++(0,1) ---++(-45:1) --(2,1);
```



また ++ ではなく + を使うと，この座標は相対指定の際に使わなくなる．

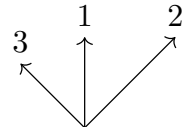
```
\tikz \draw (0,0) --(1,0) ---+(0,1) ---+(-45:1) --(2,1);
```



### 2.4.3 座標へのオプション ([1]13.1)

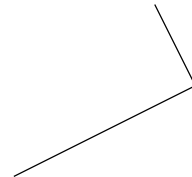
座標を使うときは ([オプション] 座標) とすることで，その (座標) に直接オプションを設定することができる．例えば次の例を参照．

```
\begin{tikzpicture}
\coordinate (a) at (0,0);
\coordinate (b) at (0,1);
\draw[->] (a) --(b) node[above]{1};
\draw[->] (a) --([xshift=1cm]b) node[above]{2};
\draw[->] (a) --([rotate=45]b) node[above]{3};
\end{tikzpicture}
```

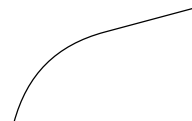


更に (座標) に設定できるオプションとして [turn] がある ([1]13.4.2)．これを設定すると，直前のパスの接線方向を  $x$  軸方向とみなした「座標」となる (原点は現在位置)．次の例を参照．

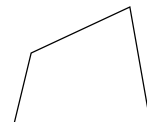
```
\tikz \draw (0,0) --(2,1) --([turn]0,1);
```



```
\tikz \draw (0,0) to[bend left=30](1,1)
--([turn]1,0);
```



```
\tikz \draw (0,0) --(1.5,0) --([turn]100:1.3cm)
--([turn]105:1.2cm) --cycle;
```



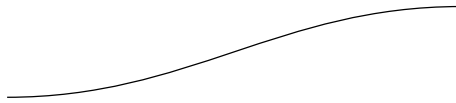
## 2.5 その他のオペレーション

この節では第1章で説明しなかったオペレーションを説明する。

### 2.5.1 Curve-To オペレーション ([1]14.3)

`..controls(座標1)and(座標2)..(座標3)` で3次ベジェ曲線を描くことができる。例えば次のようになる。

```
\tikz \draw (0,0) ..controls(2,0)and(3,1)..(5,1);
```



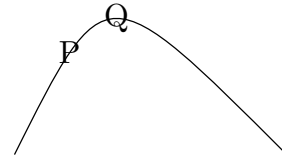
(座標<sub>1</sub>) と (座標<sub>2</sub>) が一致している場合は `and(座標2)` を省略することができる。

```
\tikz \draw (0,0) ..controls(2,1)..(5,0);
```



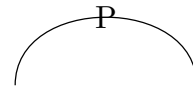
[pos] も使うことができる。

```
\tikz \draw (0,0) .. controls (1,2) .. (3,0)
node[pos=0.25]{P} node[pos=0.5]{Q};
```

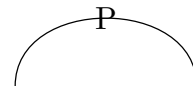


また `node` を中間に置く記法も使うことができ、この場合 `..` の直後に `node` を書く。

```
\tikz \draw (0,0) .. node{P} controls (0,1)
and (2,1) .. (2,0);
```



```
\tikz \draw (0,0) .. controls (0,1)
and (2,1) .. node{P} (2,0);
```



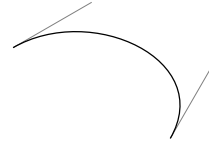
なお Curve-To オペレーションの中で `+` の記法 (第2.4.2節) を使った場合は次のように特殊な動作をする。

- (座標<sub>1</sub>) と (座標<sub>3</sub>) については、「現在位置」からの相対的な位置を表す。
- (座標<sub>2</sub>) については、(座標<sub>3</sub>) からの相対的な位置を表す。

```

\begin{tikzpicture}
\draw[help lines] (0,0) --+(30:1cm)
    ++(-30:2cm) --+(60:1cm);
\draw (0,0) .. controls +(30:1cm)
    and +(60:1cm) .. +(-30:2cm);
\end{tikzpicture}

```



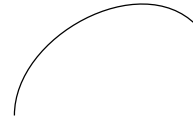
## 2.5.2 to オペレーション ([1]14.13)

今まで  $(0,0) \text{ --}(2,1)$  のような書き方をしてきたが、これは  $(0,0) \text{ to}(2,1)$  と書くこともできる。この2つの違いは、to にはオプションを付けられることである。オプションには以下のようなものがある。

- [out=角度] ([1]74.3)

パスが始点から出るときの方向を角度で指定することができる (パスは勝手に適当に曲がる)。例えば次のように [out=90] とすると、パスが始点から真上に出る。

```
\tikz \draw (0,0) to[out=90](2,1);
```



- [in=角度] ([1]74.3)

[out] と同様で、終点に入るときの角度を指定する。

```
\tikz \draw (0,0) to[out=90,in=225](2,1);
```



- [relative=真偽値] ([1]74.3)

[out] や [in] における角度の指定を「絶対的」にするか「相対的」にするかを設定する。[relative=true] の場合が「相対的」で、[relative=false] の場合が「絶対的」である。既定値は true である。

```
\tikz \draw (0,0) to[out=90,in=225,relative](2,1);
```



- [bend left=角度] ([1]74.3)

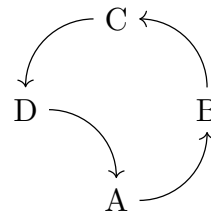
[out=角度,in=(180-角度),relative] の略記である。

```
\tikz \draw (0,0) to[bend left=70](0,2);
```



- [bend right=角度] ([1]74.3)  
[bend left] と同様.
- [bend angle=角度] ([1]74.3)  
[bend left] や [bend right] で使われる角度を設定する.

```
\begin{tikzpicture}[->,bend angle=45]
\node (a) at (1,0) {A};
\node (b) at (2,1) {B};
\node (c) at (1,2) {C};
\node (d) at (0,1) {D};
\draw (a) to[bend right] (b);
\draw (b) to[bend right] (c);
\draw (c) to[bend right] (d);
\draw (d) to[bend left] (a);
\end{tikzpicture}
```



- [looseness=値] ([1]74.3)  
「looseness」を設定する. 初期値は 1 である. 次の例を参照.

```
\begin{tikzpicture}
\draw (0,0) to[out=0,in=-90]++(1,1);
\draw (1,0) to[out=0,in=-90,
looseness=0.5]++(1,1);
\draw (2,0) to[out=0,in=-90,
looseness=2]++(1,1);
\end{tikzpicture}
```



- [out looseness=値] ([1]74.3)  
「looseness」を [out] にのみ適用する.
- [in looseness=値] ([1]74.3)  
「looseness」を [in] にのみ適用する.
- [loop] ([1]74.4)

to(座標) の (座標) は無視して「現在位置」から「現在位置」に向かうパスを定義する. この際 [looseness=8,min distance=5mm] という設定が自動的にされる. この設定の下では [in] と [out] の差を  $30^\circ$  にすると綺麗な曲線になる (と [1] は言っている). なお [loop] は [in] と [out] の後に書く必要があるようだ.

```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (2,0) {B};
\draw (a) to[loop] (b);
\draw (b) to[in=30,out=60,loop] ();
\end{tikzpicture}
```



- [loop above] ([1]74.4)  
「loop」が上に出る.
- [loop below] ([1]74.4)  
[loop above] と同様.
- [loop left] ([1]74.4)  
[loop above] と同様.
- [loop right] ([1]74.4)  
[loop above] と同様.
- [edge node={ノード}] ([1]14.13)

Line-To オペレーションと同様に, to の直後に node を書くことができる.

```
\tikz \draw (0,0) to node{数学} (2,1);
```

数学

この代わりに, edge node オプションを使って, to のオプションとしてノードを指定することができる. これは次のような形式で書く.

```
\tikz \draw (0,0)
to[edge node={node {数学}}] (2,1);
```

数学

- [edge label=文字] ([1]14.13)  
[edge node={node[auto]{文字}}] の略記である.

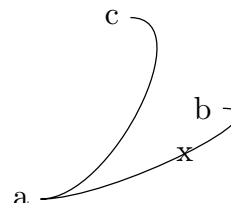
```
\tikz \draw (0,0) to[edge label=数学] (2,1);
```

数学

- [edge label'=文字] ([1]14.13)  
[edge node={node[auto,swap]{文字}}] の略記である.
- [to path=設定] ([1]14.13)

このオプションにより, to はその挙動を変えることができる. 次の例を参照.

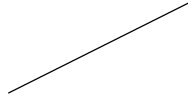
```
\begin{tikzpicture}[to path={.. controls +(1,0)
and +(1,0) .. (\tikztotarget) \tikztonodes}]
\node (a) at (0,0) {a};
\node (b) at (2,1) {b};
\node (c) at (1,2) {c};
\draw (a) to node{x} (b)
(a) to (c);
\end{tikzpicture}
```





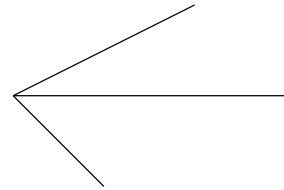
### 2.5.3 edge オペレーション ([1]17.12)

`\draw (0,0) edge(2,1);` と書くことでも線を引くことができる.



`to` との違いは「現在位置」が移動しないことである. 次の例を参照.

`\tikz \draw (0,0) edge(2,1) edge(3,0) edge(1,-1);`



### 2.5.4 arc オペレーション ([1]14.7)

`circle` オペレーションで楕円を描くことができたが, これは楕円全体が描かれてしまう. 楕円の一部のみを描きたい場合に使用するのが `arc` オペレーションである. 楕円の大きさを設定するオプションが `[x radius=長さ]` と `[y radius=長さ]` であり, 楕円のどの部分を使用するかを `[start angle=角度]` と `[end angle=角度]` で設定する. この設定した楕円弧を「現在位置」から描くことになる.

```
\tikz \draw (0,0) node{X}
      arc[x radius=1cm,y radius=5mm,
          start angle=30,end angle=180];
```



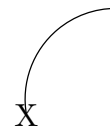
`[start angle]` と `[end angle]` の両方を設定する代わりに, この2つのどちらか片方の角度と `[delta angle]` を組み合わせることもできる.

```
\tikz \draw (0,0) (0,0) node{X}
      arc[x radius=1cm,y radius=5mm,
          start angle=30,delta angle=90];
```



また `[x radius]` と `[y radius]` が同じときは `[radius]` も使用できる.

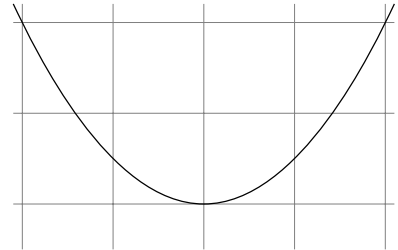
```
\tikz \draw (0,0) node{X}
      arc[radius=1cm,
          end angle=90,delta angle=-100];
```



### 2.5.5 parabola オペレーション ([1]14.9)

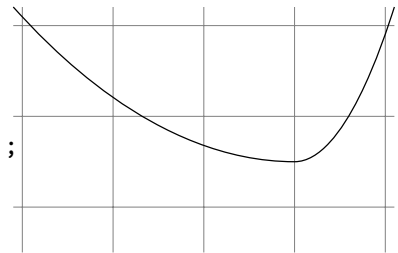
`parabola bend(座標1)(座標2)` を `parabola` オペレーションといい、「現在位置」から (座標<sub>2</sub>) に向かって放物線を描く。このとき放物線は (座標<sub>1</sub>) を頂点にするように描かれる。

```
\begin{tikzpicture}
\draw[help lines] (-2.1,-0.5) grid(2.1,2.205);
\draw (-2.1,2.205) parabola bend(0,0)(2.1,2.205);
\end{tikzpicture}
```



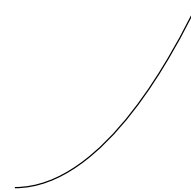
ただし放物線にならないような座標の与え方をすると放物線にならないため注意が必要。

```
\begin{tikzpicture}
\draw[help lines] (-2.1,-0.5) grid(2.1,2.205);
\draw (-2.1,2.205) parabola bend(1,0.5)(2.1,2.205);
\end{tikzpicture}
```



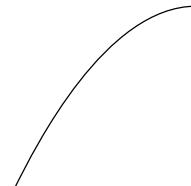
`bend(座標1)` はオプションを使って `parabola[bend=(座標1)](座標2)` という形式で設定することもできる。また `[bend]` は省略することもできて、その場合 `[bend]` には「現在位置」を設定した状態となる。

```
\tikz \draw (0,0) parabola (2,2);
```



(座標<sub>2</sub>) を `[bend]` に設定したい場合は `[bend at end]` を使用することもできる。

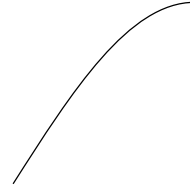
```
\tikz \draw (0,0) parabola[bend at end] (2,2);
```



## 2.5.6 sine オペレーション ([1]14.10)

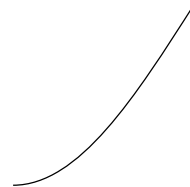
`sin(座標)` を sine オペレーションといい, 「現在位置」から (座標) に向かって正弦曲線を描く. このとき正弦曲線は  $0 \sim \pi/2$  の範囲が使用される.

```
\tikz \draw (0,0) sin(2,2);
```



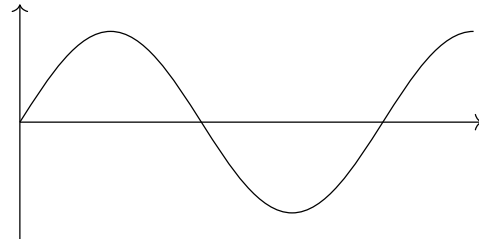
同様に余弦曲線も `cos(座標)` で描くことができる.

```
\tikz \draw (0,0) cos(2,2);
```



長い正弦曲線を描きたいときは `sin` と `cos` を組み合わせる.

```
\begin{tikzpicture}
\draw[>-] (0,-1.3) --(0,1.3);
\draw[>-] (0,0) --(5,0);
\draw (0,0) sin(1,1) cos(2,0) sin(3,-1)
      cos(4,0) sin(5,1);
\end{tikzpicture}
```



## 2.5.7 decorate オペレーション ([1]24, 50)

`\usetikzlibrary{decorations}` を使うと `decorate` オペレーションによりパスの形を色々なものにすることができる. これは `decorate{オペレーション}` という形で使用し, オペレーション で設定したパスに対して「装飾」を行う. どのような「装飾」をするかは `decoration` オプションで設定する. 例えば次のようになる (この例は `\usetikzlibrary{decorations.pathmorphing}` が必要).

```
\tikz \draw decorate[decoration={name=zigzag}]
{ (0,0) rectangle(2,1) };
```



この例のように `name` に色々な名前を指定することで「装飾」の形を設定できる (`name` は省略可能). 名前には次のようなものがある.

- `curveto` ([1]50.3.2)  
元のパスに沿った線を引く.
- `lineto` ([1]50.3.1)  
直線を引く.
- `moveto` ([1]50.4)  
何も描かず移動だけする. 主に後で説明する `pre` や `post` と組み合わせて使用する.

次は使用するのに `\usetikzlibrary{decorations.pathmorphing}` が必要.

- `zigzag` ([1]50.3.1)  
ジグザグ線を引く. `amplitude` と `segment length` でジグザグの大きさを設定できる.

```
\tikz \draw decorate[decoration={name=zigzag,
    amplitude=2mm,segment length=4mm}]
    { (0,0) --(2,0) };
```



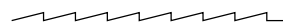
- `snake` ([1]50.3.2)  
正弦曲線を引く. `amplitude` と `segment length` で波の大きさを設定できる.

```
\tikz \draw decorate[decoration=snake]
    { (0,0) --(3,0) };
```



- `saw` ([1]50.3.1)  
のこぎり状の線を引く. `amplitude` と `segment length` でのこぎりの大きさを設定できる.

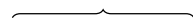
```
\tikz \draw decorate[decoration=saw]
    { (0,0) --(3,0) };
```



次は使用するのに `\usetikzlibrary{decorations.pathreplacing}` が必要.

- `brace` ([1]50.4)  
中括弧の形で線を引く. `amplitude` と `aspect` で中括弧の大きさを設定できる.

```
\tikz \draw decorate[decoration=brace]
    { (0,0) --(2,0) };
```

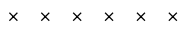


次は使用するのに `\usetikzlibrary{decorations.shapes}` が必要.

- `crosses` ([1]50.6.3)  
一定の間隔でバツ印を描く. 間隔の長さは `segment length` で, バツ印の大きさ

は `shape width` と `shape height` で設定できる.

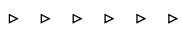
```
\tikz \draw decorate[decoration=crosses]
{ (0,0) --(2,0) };
```



- `triangles` ([1]50.6.3)

一定の間隔で三角形を描く. 間隔の長さは `segment length` で, 三角形の大きさは `shape width` と `shape height` で設定できる.

```
\tikz \draw decorate[decoration=triangles]
{ (0,0) --(2,0) };
```



次は使用するのに `\usetikzlibrary{decorations.text}` が必要.

- `text along path` ([1]50.7)

パスに沿って文字を書く. 詳しくは [1]50.7 を参照.

`\path decorate[decoration={設定}]{オペレーション}` は `decorate` オプションを使って `\path[decorate,decoration={設定}] オペレーション` という形式で書くこともできる. また `[decoration]` には `name` 以外にも次のようなものを設定できる.

- `raise=長さ` ([1]24.4.1)

パスの進行方向に向かって左側に装飾をずらす. 長さ に負の値を設定した場合は右にずらす.


- `pre=装飾名` ([1]24.4.1)

パスの開始部分に適用する装飾を指定する.

- `pre length=長さ` ([1]24.4.1)

`pre` での設定を使用する長さを指定する. 初期値は `0pt` である.

```
\tikz \draw decorate[decoration={snake,
pre=lineto,pre length=1cm}]
{ (0,0) --(3,0) };
```



- `post` ([1]24.4.1)

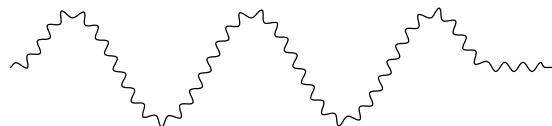
`pre` と同様.

- `post length=長さ` ([1]24.4.1)

`pre length` と同様.

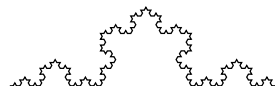
`decorate` オペレーションは重ねて使うこともできる.

```
\tikz \draw
  decorate[decoration={snake,amplitude=0.5mm,segment length=2mm}]
  { decorate[decoration={snake,amplitude=6mm,segment length=2cm}]
    {(0,0) --(6,0)} };
```



これを使ってフラクタルを描くのが `\usetikzlibrary{decorations.fractals}` である ([1]50.8).

```
\tikz \draw[decoration=Koch snowflake]
  decorate { decorate { decorate { decorate
    { (0,0) --(3,0) } } } };
```

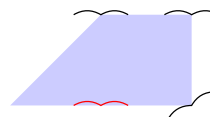


decoration については自分で定義することもできる。詳しくは [1]103 を参照。

## 2.5.8 pic オペレーション ([1]18)

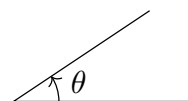
次のように [名称/.pic=設定] と記述すると, `pic{名称}` により「現在位置」に設定に書いたものを描くことができる (詳しくは [1]18 を参照)。

```
\begin{tikzpicture}[seagull/.pic={
  \draw (-3mm,0) to[bend left] (0,0) to[bend left] (3mm,0);
}]
\node at (0,3.5) {};
\fill [fill=blue!20] (1,1)
  -- (2,2) pic{seagull}
  -- (3,2) pic{seagull}
  -- (3,1) pic[rotate=30]{seagull}
  -- (2,1) pic[red]{seagull};
\end{tikzpicture}
```



pic オペレーションの応用例として `angle` がある (これは `\usetikzlibrary{angles}` が必要)。これを使うと角度の表示ができる (詳しくは [1]41 を参照)。

```
\begin{tikzpicture}
\coordinate (A) at (2,0);
\coordinate (B) at (0,0);
\coordinate (C) at (1.5,1);
\draw (A) --(B) --(C)
  pic[draw,->,angle eccentricity=1.5,
    pic text={\theta}]{angle=A--B--C};
\end{tikzpicture}
```



```
\begin{tikzpicture}
\coordinate (A) at (1.5,0);
\coordinate (B) at (0,0);
\coordinate (C) at (0,1);
\draw (A) --(B) --(C)
  pic[draw,angle radius=2mm]{right angle=A--B--C};
\end{tikzpicture}
```



また `\path pic` の略記として `\pic` が使用できる ([1]18.2).

## 2.5.9 plot オペレーション ([1]22)

`plot` オペレーションにより, データをプロットしてグラフを書くことができる. まず基本的な使い方として `plot coordinates{複数座標}` というものがある. これは設定した座標を結んだものを表す.

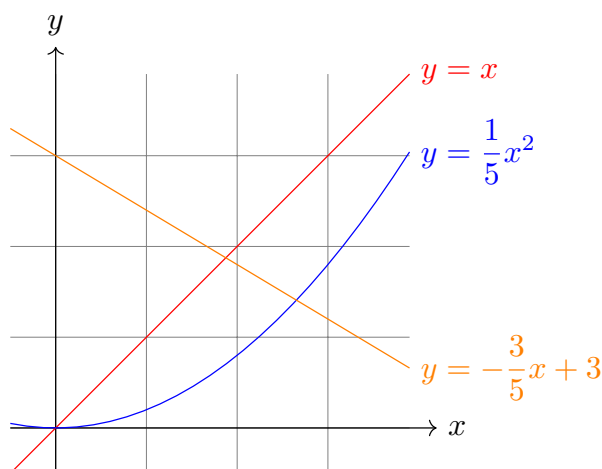
```
\tikz \draw plot coordinates
      {(0,0)(1,1)(2,1)(3,0)(3,1)};
```



この座標データは `plot file{ファイル名}` により外部ファイルから取り込むこともできるので, 例えば実験データを取り込んでグラフで表示するといったこともできる (ファイルの形式については [1]22.4 を参照).

また `plot(座標の計算式)` と書くと, 計算した座標を使ってグラフを書くことができる (第 2.11 節も参照). 座標の計算式 には変数として `\x` を使うことができ, `\x` の動く範囲は `[domain=開始: 終了]` という形式で設定できる.

```
\begin{tikzpicture}[domain=-0.5:3.9]
\draw[help lines] (-0.5,-0.5) grid (3.9,3.9);
\draw[->] (-0.5,0) -- (4.2,0) node[right] {$x$};
\draw[->] (0,-0.5) -- (0,4.2) node[above] {$y$};
\draw[color=red] plot(\x,\x) node[right] {$y=x$};
\draw[color=blue] plot(\x,\x*\x/5) node[right] {$y=\frac{1}{5}x^2$};
\draw[color=orange] plot(\x,-\x*3/5+3) node[right] {$y=-\frac{3}{5}x+3$};
\end{tikzpicture}
```

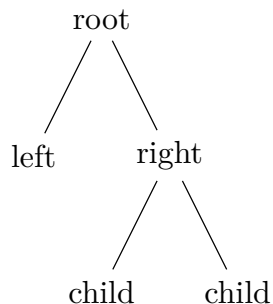


詳しくは [1]22 を参照. また [1]82 も参照.

## 2.5.10 child オペレーション ([1]21)

木構造を簡単に描くために child オペレーションというものが用意されている。

```
\begin{tikzpicture}
\node {root}
  child { node{left} }
  child { node{right}
    child { node{child} }
    child { node{child} }
  };
\end{tikzpicture}
```

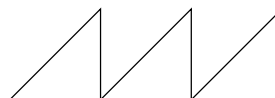


詳しくは [1]21 を参照。またライブラリとしては trees ライブラリというものもある ([1]76)。

## 2.5.11 foreach オペレーション ([1]14.14)

foreach オペレーションというオペレーションを使うことで繰り返し処理をすることができる。これは `foreach 変数 in {動く範囲} {オペレーション}` という形式で書く。例えば次のようになる。

```
\tikz \draw (0,0) foreach \x in {1,...,3}
{ --(\x,1) --(\x,0) };
```



foreach オペレーションは `\path` の中で使用するオペレーションだが、tikzpicture 環境とは関係なく使えるコマンドとして `\foreach` というコマンドも TikZ には用意されている ([1]88)。これは例えば `$\foreach \x in {a,...,e} {\x}` と書くと `[a][b][c][d][e]` となる。

## 2.5.12 let オペレーション ([1]14.15)

`\usetikzlibrary{calc}` とすることで let オペレーションを使用することができる。これは let 式 in の形で書き、例えば次のようなことができる。

```
\tikz \draw let \p1 = (0,1), \p2 = (1,1) in
(0,0) --(\p1) --(\p2) --cycle;
```



式の部分には `変数 = 値` のような式をカンマ区切りで 1 つ以上書く。変数 には次のようなものを使用できる。

- `\p` 点レジスター ([1]14.15)

これは「座標」を表す変数である。点レジスター の部分には、上記の例のように数



字 1 文字を使える他, { } を使うことで文字も使用することができる.

```
\tikz \draw let \p{foo} = (0,1), \p{bar} = (1,1) in
(0,0) --(\p{foo}) --(\p{bar}) --cycle;
```



\p 点レジスター の  $x$  座標,  $y$  座標はそれぞれ \x 点レジスター, \y 点レジスターで取り出すことができる.

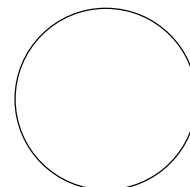
```
\tikz \draw let \p1 = (1,1) in
(0,0) --(\x1,0) --(0,\y1) --cycle;
```



- \n 数レジスター ([1]14.15)

これは数値や長さを表す変数である. 数レジスター の部分は 点レジスター と同様である.

```
\tikz \draw let \p1 = (1,1) , \n1 = 1cm in
(\p1) circle[radius=\n1];
```



変数 は let オペレーションの中でしか使用できないので, 外から参照したい場合は例えば coordinate 等で名前を付けておくとよい.

```
\begin{tikzpicture}
\path let \p1 = (0,1), \p2 = (1,1) in
coordinate (x) at (\p1)
coordinate (y) at (\p2);
\draw (0,0) --(x) --(y) --cycle;
\end{tikzpicture}
```



## 2.6 label オプションと pin オプション

これらは node に指定するオプションであり, 別の新たなノードを追加することができる.

### 2.6.1 label オプション ([1]17.10.2)

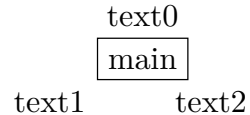
まず label オプションについては [label=位置名: 文字] の形式で指定する. これは位置名 で指定した場所に「文字 を [node contents] として持つノード」を追加する. (位置名 は north, above のような名称か, もしくは角度で指定する. )

```
\tikz \node[draw,label=above:text] at (0,0) {main};
```




[label] は複数指定することも可能である.

```
\tikz \node[draw,label=above:text0,
            label=south west:text1,
            label=-45:text2] at (0,0) {main};
```




また 位置名 の前に [ ] で指定することで, 追加されるノードにオプションを付けることができる.

```
\tikz \node[draw,label={[fill=red!20]above:text}]
        at (0,0) {main};
```



また 位置名: の部分は省略することができ, その場合既定値が使われる. 既定値は通常は above である (これは以下で述べる [label position] で変更が可能).

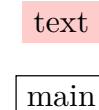
```
\tikz \node[draw,label={[fill=red!20]text}] at (0,0) {main};
```



- [label distance=長さ] ([1]17.10.2)

元のノードと新しく作られるノードの間の長さを設定する. 初期値は 0pt である.

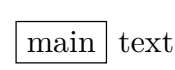
```
\tikz \node[draw,label={[fill=red!20,
            label distance=10pt]above:text}]
        at (0,0) {main};
```



- [label position=位置名] ([1]17.10.2)

[label] で指定する 位置名 の既定値を設定する. 初期値は above である.

```
\tikz[label position=right]
        \node[draw,label=text] at (0,0) {main};
```



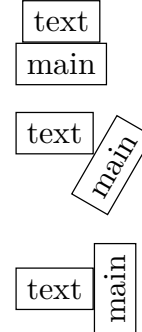
- [absolute=真偽値] ([1]17.10.2)

位置名での指定について, 絶対的か相対的かを設定する. 既定値は true である. 例えば, まず通常の場合, ノードを [rotate] によって回転させると, [label] により作られるノードの位置も回転していく.




```
\tikz \node[draw,label={[draw]text}]
        at (0,0) {main};

\tikz \node[draw,rotate=60,
            label={[draw]text}]
        at (0,0) {main};

\tikz \node[draw,rotate=90,
            label={[draw]text}]
        at (0,0) {main};
```

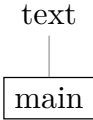


[absolute=true] を設定した場合、回転には影響されなくなる。

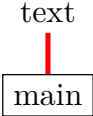
|                                                                                                                       |                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <pre>\tikz \node[draw,             label={[draw,absolute]text}]             at (0,0) {main};</pre>                    |  |
| <pre>\tikz \node[draw,rotate=60,absolute,             label={[draw,absolute]text}]             at (0,0) {main};</pre> |  |
| <pre>\tikz \node[draw,rotate=90,             label={[draw,absolute]text}]             at (0,0) {main};</pre>          |  |

## 2.6.2 pin オプション ([1]17.10.3)

pin オプションも label オプションと同様で、[pin=位置名: 文字] の形式で指定する。[label] との違いは、[pin] は元のノードと新しいノードの間に線が引かれることである。

|                                                              |                                                                                       |
|--------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <pre>\tikz \node[draw,pin=above:text] at (0,0) {main};</pre> |  |
|--------------------------------------------------------------|---------------------------------------------------------------------------------------|


- [pin distance=長さ] ([1]17.10.3)  
[label distance] の [pin] バージョンである。初期値は 3ex である。
- [pin position=位置名] ([1]17.10.2)  
[label position] の [pin] バージョンである。初期値は above である。
- [pin edge={設定}] ([1]17.10.2)  
[pin] で引かれる線の設定を行う。

|                                                                                                               |                                                                                       |
|---------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <pre>\tikz \node[draw,             pin={[pin edge={red,ultra thick}]text}]             at (0,0) {main};</pre> |  |
|---------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|

### 2.6.3 The Quotes Syntax ([1]17.10.4)

`\usetikzlibrary{quotes}` とすることで `[label]` と `[pin]` について " を使った略記をすることが出来るようになる. 詳しくは [1]17.10.4 を参照.

```
\tikz \node[draw,"text" above] at (0,0) {main};
```



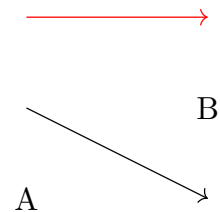
## 2.7 perpendicular 座標系 ([1]13.3.1)

この座標系は

```
(perpendicular cs:vertical line through={ (座標1) },  
horizontal line through={ (座標2) })
```

の形で書き「 $x$  座標が (座標<sub>1</sub>) と同じ」で「 $y$  座標が (座標<sub>2</sub>) と同じ」となる「座標」を表す. これも略記法があり (座標<sub>1</sub>|座標<sub>2</sub>) もしくは (座標<sub>2</sub>-|座標<sub>1</sub>) と書くことができる. 例えば次のようになる.

```
\begin{tikzpicture}  
  \node (a) at (0,0) {A};  
  \node (b) at (2,1) {B};  
  \draw[->] (a|-b) --(a|b);  
  \draw[->,red] (a|-0,2) --(b|-0,2);  
\end{tikzpicture}
```



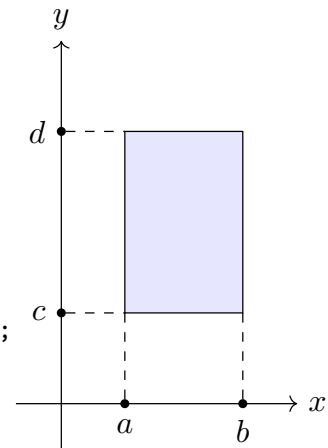
なので例えば,  $xy$  平面上に閉区間  $[a, b]$  と  $[c, d]$  の直積は次のように描くことができる.

```
\begin{tikzpicture}[point/.style={fill,
    shape=circle,inner sep=1pt,outer sep=0pt}]
% x 軸 y 軸
\draw[->] (-0.5,0) --(2.6,0) node[right]{$x$};
\draw[->] (0,-0.5) --(0,4) node[above]{$y$};

% 軸上の点 (値は label で表示)
\node[point,label=below:$a$] (x0) at (0.7,0) {};
\node[point,label=below:$b$] (x1) at (2,0) {};
\node[point,label=left:$c$] (y0) at (0,1) {};
\node[point,label=left:$d$] (y1) at (0,3) {};

% 直積
\draw[fill=blue!10] (y0 -| x0) rectangle(y1 -| x1);

% 点線
\draw[dashed] (x0) --(x0 |- y0);
\draw[dashed] (x1) --(x1 |- y0);
\draw[dashed] (y0) --(x0 |- y0);
\draw[dashed] (y1) --(x0 |- y1);
\end{tikzpicture}
```

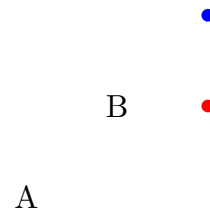


## 2.8 座標計算 ([1]13.5)

`\usetikzlibrary{calc}` とすることで座標計算をすることが出来るようになる. 座標計算は ( ) の中で  $\$$  を使って ( $\$$ 座標の計算式 $\$$ ) という形で書く. 行える計算については次のようなものがある.

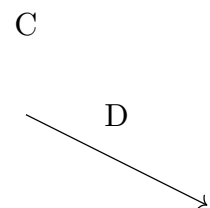
- 座標に対して  $+$ ,  $-$  や  $*$  で計算ができる. 例えば次のようになる.

```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (1,1) {B};
\fill[red] ($ (b)+(1,0) $) circle[radius=2pt];
\fill[blue] ($ 2*(b) $) circle[radius=2pt];
\end{tikzpicture}
```



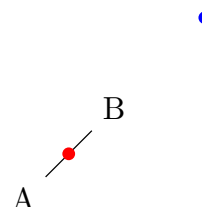
入れ子にすることでより複雑な計算もできる.

```
\begin{tikzpicture}
\node (c) at (0,1) {C};
\node (d) at (1,0) {D};
\draw[->] (0,0) --($ 2*($ (d)-($ (c)) $)+(0,1) $);
\end{tikzpicture}
```



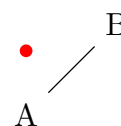
- $(\text{座標}_1)!\text{割合}!(\text{座標}_2)$  と書くと、線分  $(\text{座標}_1) -- (\text{座標}_2)$  の内分点や外分点の座標を計算することができる。割合 には内分や外分の位置を設定する。例えば 0 なら  $(\text{座標}_1)$  を、1 なら  $(\text{座標}_2)$  を、0.5 なら中点を表す。

```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (1,1) {B};
\draw (a) --(b);
\fill[red] ($(a)!0.5!(b)$) circle[radius=2pt];
\fill[blue] ($(a)!2!(b)$) circle[radius=2pt];
\end{tikzpicture}
```



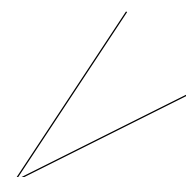
また  $(\text{座標}_2)$  の直前には 角度: と付けることもできる。これを付けると  $(\text{座標}_2)$  を、 $(\text{座標}_1)$  を中心に 角度 だけ回転させて計算する。

```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (1,1) {B};
\draw (a) --(b);
\fill[red] ($(a)!0.5!45:(b)$) circle[radius=2pt];
\end{tikzpicture}
```



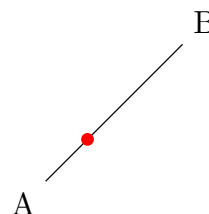
これを使うと指定した角度で線を引くことができる。

```
\begin{tikzpicture}
\coordinate (a) at (0,0);
\coordinate (b) at (2,1);
\draw (b) --(a) --($ (a)!1!30:(b) $);
\end{tikzpicture}
```



- $(\text{座標}_1)!\text{長さ}!(\text{座標}_2)$  と書くと、線分  $(\text{座標}_1) -- (\text{座標}_2)$  上の点で、 $(\text{座標}_1)$  から 長さ の位置にある座標を計算することができる。

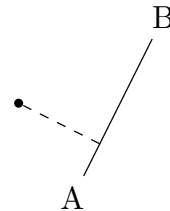
```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (2,2) {B};
\draw (a) --(b);
\fill[red] ($(a)!1cm!(b)$) circle[radius=2pt];
\end{tikzpicture}
```



この場合も  $(\text{座標}_2)$  の直前に 角度: と付けることができる。これを使うと、直線

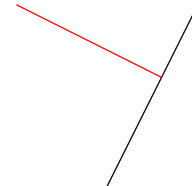
上の点から向かって左に 1 cm の位置を計算したりできる。

```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (1,2) {B};
\draw (a) --(b);
\coordinate (c) at ($(a)!0.3!(b)$);
\node[fill,shape=circle,inner sep=1pt]
(x) at ($(c)!1cm!90:(b)$) {};
\draw[dashed] (c) --(x);
\end{tikzpicture}
```



- $(\text{座標}_1)!(\text{座標}_2)!(\text{座標}_3)$  と書くと,  $(\text{座標}_2)$  から線分  $(\text{座標}_1) -- (\text{座標}_3)$  に対して垂線を下したときの足の座標を計算することができる。

```
\begin{tikzpicture}
\coordinate (a) at (0, 2);
\coordinate (b) at (1, 0);
\coordinate (c) at (2, 2);
\draw (b) --(c);
\draw[red] (a) --($ (b)!(a)!(c) $);
\end{tikzpicture}
```



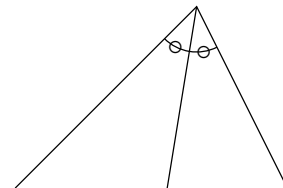
例えば角の二等分線であれば次のように描くことができる。

```
\begin{tikzpicture}
% 3 点
\coordinate (A) at (2,2);
\coordinate (B) at (0,0);
\coordinate (C) at (3,0);
\draw (B) --(A) --(C);

% 二等分する座標を求める
\coordinate (P0) at ($(A)!1cm!(B)$);
\coordinate (P1) at ($(A)!1cm!(C)$);
\coordinate (P) at ($(P0)!0.5!(P1)$);

% 二等分線 (適当に伸ばす)
\draw (A) --($ (A)!2.5!(P) $);

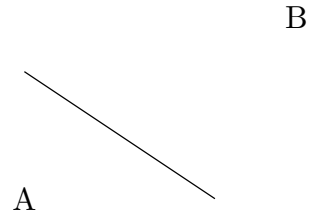
% 等しい記号 (pic オペレーション)
\pic[draw,pic text={\circ$},angle eccentricity=1.03]{angle=B--A--P};
\pic[draw,pic text={\circ$},angle eccentricity=1.03]{angle=P--A--C};
\end{tikzpicture}
```



座標計算と `perpendicular` 座標系を組み合わせる場合は `{ }` を使用する.

```
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (3,2) {B};

\draw ({$(a)!0.7!(b)$} |- a)
      --({$(a)!0.7!(b)$} -| a);
\end{tikzpicture}
```



## 2.9 tangent 座標系 ([1]13.3.1)

座標系名 `tangent` を使うと, 1 点から円に接線を引いたときの接点の座標を計算することができる (使うには `\usetikzlibrary{calc}` が必要). この座標系名は

`(tangent cs:node=ノード名,point={ (座標) },solution=番号)`

の形で使用する. ノード名 には円となるノードの名前を設定する (このとき対象のノードは `[shape=circle]` にしておく). (座標) には接線を引く点の「座標」を設定する. こ



のような接線は 2 本あるから 番号 でどちらを参照するか指定する．次の例を参照．

```
% この例は \usetikzlibrary{angles} も必要
\begin{tikzpicture}[point/.style={fill,
    shape=circle,inner sep=1pt,outer sep=0pt}]
% 点 P
\node[point,label=right:$P$] (P) at (3,0.6) {};

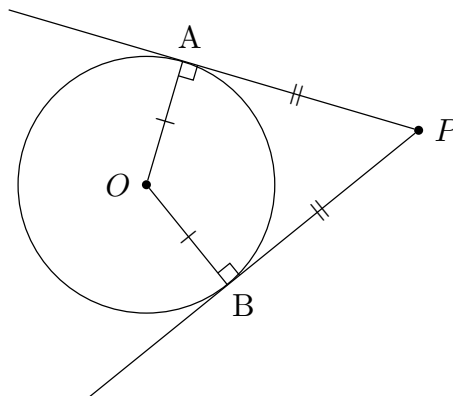
% 円 O
\node[point,label=left:$O$] (O) at (0,0) {};
\node[draw,shape=circle,inner sep=1cm] (circle) at (O) {};

% 接線 ([turn] で線を伸ばす)
\draw (P) --(tangent cs:node=circle,point={P},solution=1)
    coordinate (A) --([turn]2,0);
\draw (P) --(tangent cs:node=circle,point={P},solution=2)
    coordinate (B) --([turn]2,0);

% 垂線
\draw (O) --(A);
\draw (O) --(B);
\node at ($(O)!17mm!(A)$) {A};
\node at ($(O)!17mm!(B)$) {B};

% 直角の記号 (pic オペレーション)
\pic[draw,angle radius=1.7mm]{right angle=O--A--P};
\pic[draw,angle radius=1.7mm]{right angle=O--B--P};

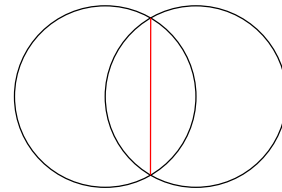
% 等しい記号
\path (O) -- node[sloped,rotate=90]{$-$} (A);
\path (O) -- node[sloped,rotate=90]{$-$} (B);
\path (P) -- node[sloped,rotate=90]{$=$} (A);
\path (P) -- node[sloped,rotate=90]{$=$} (B);
\end{tikzpicture}
```



## 2.10 パスの交点を参照する ([1]13.3.2)

`\usetikzlibrary{intersections}` とすることで、パスの交点の座標を参照することができる。交点を計算したいパスに `[name path=名前]` で名前を付けて、交点を参照したいパスに `[name intersections={of 名前1 and 名前2}]` と書く。こうすると交点の座標を (intersection-番号) で参照できるようになる (一般に交点は複数あるため、それを 番号 で区別する)。

```
\begin{tikzpicture}
\draw[name path=c1] (0,0) circle[radius=1cm];
\draw[name path=c2] (1,0) circle[radius=1cm];
\draw[red,name intersections={of=c1 and c2}]
(intersection-1) --(intersection-2);
\end{tikzpicture}
```



例えば三角形の垂心は次のように描くことができる (垂線の足の計算方法は第 2.8 節を

参照).

```
% この例は \usetikzlibrary{angles} も必要
\begin{tikzpicture}[point/.style={fill,
    shape=circle,inner sep=1pt,outer sep=0pt}]
% 3点
\coordinate (A) at (2,2);
\coordinate (B) at (0,0);
\coordinate (C) at (3,0);

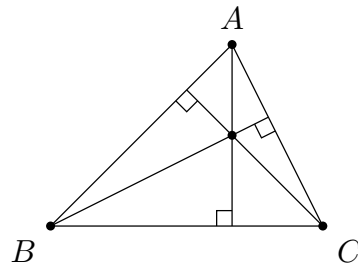
% 三角形
\draw (A) --(B) --(C) --cycle;
\node[point,label=90 :$A$] at (A) {};
\node[point,label=225:$B$] at (B) {};
\node[point,label=-45:$C$] at (C) {};

% 垂線の足
\coordinate (x) at ($(B)!(A)!(C)$);
\coordinate (y) at ($(C)!(B)!(A)$);
\coordinate (z) at ($(A)!(C)!(B)$);

% 垂線 (A からの垂線と B からの垂線に名前を付ける)
\draw[name path=line1] (A) --(x);
\draw[name path=line2] (B) --(y);
\draw (C) --(z);

% 直角の記号 (pic オペレーション)
\pic[draw,angle radius=1.7mm]{right angle=A--x--B};
\pic[draw,angle radius=1.7mm]{right angle=B--y--C};
\pic[draw,angle radius=1.7mm]{right angle=C--z--B};

% 垂心 (A からの垂線と B からの垂線の交点に丸を付ける)
\node[point,name intersections={of=line1 and line2}]
    at (intersection-1) {};
\end{tikzpicture}
```



## 2.11 計算式について ([1]94)

第 1.2 節でも説明した通り，座標を指定する際などいくつかの場所では計算式を書くことができる．このような場所では + などの他に色々な関数も書くことができる．例えば次のように三角形の外接円の半径を `vecLen` で計算することができる (中点，垂直二等分

線の描き方は第 2.8 節を参照. 外心の座標の計算方法は第 2.10 節を参照).

```
% この例は \usetikzlibrary{angles} と
% \usetikzlibrary{intersections} が必要
\begin{tikzpicture}[point/.style={fill,
    shape=circle,inner sep=1pt,outer sep=0pt}]
% 3 点
\coordinate (A) at (2,2);
\coordinate (B) at (0,0);
\coordinate (C) at (3,0);

% 三角形
\draw (A) --(B) --(C) --cycle;
\node[point,label=90 :$A$] at (A) {};
\node[point,label=225:$B$] at (B) {};
\node[point,label=-45:$C$] at (C) {};

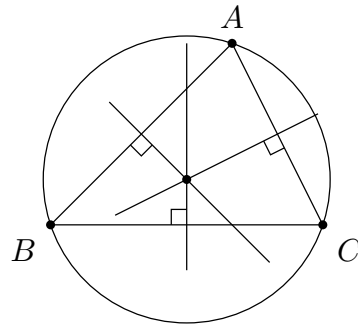
% 中点を計算する
\coordinate (x) at ($(B)!0.5!(C)$);
\coordinate (y) at ($(C)!0.5!(A)$);
\coordinate (z) at ($(A)!0.5!(B)$);

% 垂直二等分線を描いて名前を付ける
\draw[name path=line1] ($(x)!5mm!90:(B)$) --(x) --([turn]2,0);
\draw[name path=line2] ($(y)!5mm!90:(C)$) --(y) --([turn]2,0);
\draw
    ($ (z)!5mm!90:(A)$) --(z) --([turn]2,0);

% 外心
\node[point,name intersections={of=line1 and line2}]
    (O) at (intersection-1) {};

% 直角の記号 (pic オペレーション)
\pic[draw,angle radius=1.7mm]{right angle=0--x--B};
\pic[draw,angle radius=1.7mm]{right angle=0--y--C};
\pic[draw,angle radius=1.7mm]{right angle=0--z--B};

% 外接円の半径を計算する
\draw let \p1 = (O), \n1 = {veclen(\x1,\y1)} in (O) circle[radius=\n1];
\end{tikzpicture}
```



使用できる関数には次のようなものがある<sup>\*3</sup>.

- `add(x,y)` ([1]94.2, 94.3.1)  
 $x$  と  $y$  の和を計算する. 略記として  $x+y$  と書くこともできる.

<sup>\*3</sup> T<sub>E</sub>X の仕様上  $\pm 16383.99999$  を超える値は扱えないようなので注意が必要.

- `subtract(x,y)` ([1]94.2, 94.3.1)  
x と y の差を計算する. 略記として `x-y` と書くこともできる.
- `neg(x)` ([1]94.2, 94.3.1)  
x の  $-1$  倍を計算する. 略記として `-x` と書くこともできる.
- `multiply(x,y)` ([1]94.2, 94.3.1)  
x と y の積を計算する. 略記として `x*y` と書くこともできる.
- `divide(x,y)` ([1]94.2, 94.3.1)  
x と y の商 (の整数部分) を計算する. 略記として `x/y` と書くこともできる.
- `e` ([1]94.3.1)  
2.718281828 を表す.
- `pi` ([1]94.3.4)  
3.141592654 を表す.
- `factorial(x)` ([1]94.2, 94.3.1)  
x の階乗を計算する. 略記として `x!` と書くこともできる.
- `sqrt(x)` ([1]94.3.1)  
x の正の平方根を計算する.
- `pow(x)` ([1]94.2, 94.3.1)  
x の y 乗を計算する. 略記として `x^y` と書くこともできる.
- `exp(x)` ([1]94.3.1)  
e の x 乗を計算する.
- `ln(x)` ([1]94.3.1)  
x の自然対数を計算する.
- `log10(x)` ([1]94.3.1)  
x の常用対数を計算する.
- `log2(x)` ([1]94.3.1)  
x の二進対数を計算する.
- `abs(x)` ([1]94.3.1)  
x の絶対値を計算する.
- `mod(x,y)` ([1]94.3.1)  
x を y で割った余りを計算する (符号は `x/y` と同じものを使用する).
- `Mod(x,y)` ([1]94.3.1)  
x を y で割った余りを計算する. こちらは常に 0 以上の値を返す.
- `sign(x)` ([1]94.3.1)

- $x$  の正負に応じて  $-1$  か  $0$  か  $1$  を返す.
- `round(x)` ([1]94.3.2)
  - $x$  に近い整数を返す.
- `floor(x)` ([1]94.3.2)
  - $x$  を超えない最大の整数を返す.
- `ceil(x)` ([1]94.3.2)
  - $x$  以上の最小の整数を返す.
- `int(x)` ([1]94.3.2)
  - $x$  の整数部分を返す.
- `frac(x)` ([1]94.3.2)
  - $x$  の小数部分を返す.
- `gcd(x,y)` ([1]94.3.3)
  - $x$  と  $y$  の最大公約数を計算する.
- `isodd(x)` ([1]94.3.3)
  - $x$  の整数部分の偶奇に応じて  $0$  か  $1$  を返す.
- `iseven(x)` ([1]94.3.3)
  - $x$  の整数部分の偶奇に応じて  $1$  か  $0$  を返す.
- `isprime(x)` ([1]94.3.3)
  - $x$  の整数部分が素数なら  $1$  , そうでなければ  $0$  を返す.
- `rad(x)` ([1]94.3.4)
  - $x$  度を弧度法で表した時の値を返す.
- `deg(x)` ([1]94.2, 94.3.4)
  - $x$  ラジアンを度数法で表した時の値を返す. 略記として `xr` と書くこともできる.
- `sin(x)` ([1]94.3.4)
  - $x$  度の正弦を計算する.
- `cos(x)` ([1]94.3.4)
  - $x$  度の余弦を計算する.
- `tan(x)` ([1]94.3.4)
  - $x$  度の正接を計算する.
- `sec(x)` ([1]94.3.4)
  - $x$  度の正割を計算する.
- `cosec(x)` ([1]94.3.4)
  - $x$  度の余割を計算する.

- `cot(x)` ([1]94.3.4)  
x 度の余接を計算する.
- `asin(x)` ([1]94.3.4)  
逆正弦関数を計算する. 値は度数法で  $-90 \sim 90$  で返される.
- `acos(x)` ([1]94.3.4)  
逆余弦関数を計算する. 値は度数法で  $0 \sim 180$  で返される.
- `atan(x)` ([1]94.3.4)  
逆正接関数を計算する. 値は度数法で返される.
- `atan2(y,x)` ([1]94.3.4)  
逆正接関数を計算する. 値は度数法で返される.
- `sinh(x)` ([1]94.3.8)  
x の双曲線正弦を計算する.
- `cosh(x)` ([1]94.3.8)  
x の双曲線余弦を計算する.
- `tanh(x)` ([1]94.3.8)  
x の双曲線正接を計算する.
- `equal(x,y)` ([1]94.2, 94.3.5)  
x と y が等しいなら 1, そうでないなら 0 を返す. 略記として `x==y` と書くこともできる.
- `greater(x,y)` ([1]94.2, 94.3.5)  
x が y より大きいなら 1, そうでないなら 0 を返す. 略記として `x>y` と書くこともできる.
- `less(x,y)` ([1]94.2, 94.3.5)  
x が y より小さいなら 1, そうでないなら 0 を返す. 略記として `x<y` と書くこともできる.
- `notequal(x,y)` ([1]94.2, 94.3.5)  
x と y が等しくないなら 1, そうでないなら 0 を返す. 略記として `x!=y` と書くこともできる.
- `notgreater(x,y)` ([1]94.2, 94.3.5)  
x が y 以下なら 1, そうでないなら 0 を返す. 略記として `x<=y` と書くこともできる.
- `notless(x,y)` ([1]94.2, 94.3.5)  
x が y 以上なら 1, そうでないなら 0 を返す. 略記として `x>=y` と書くこともできる.

きる.

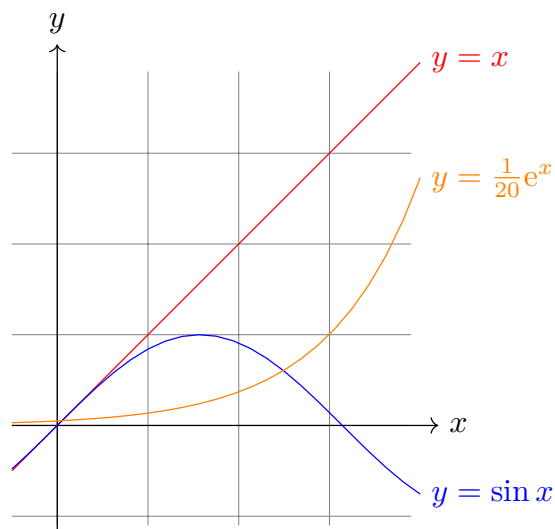
- `and(x,y)` ([1]94.2, 94.3.5)  
x と y が両方とも 0 以外なら 1, そうでないなら 0 を返す. 略記として `x&&y` と書くこともできる.
- `or(x,y)` ([1]94.2, 94.3.5)  
x と y の少なくとも一方が 0 以外なら 1, そうでないなら 0 を返す. 略記として `x||y` と書くこともできる.
- `not(x)` ([1]94.2, 94.3.5)  
x が 0 なら 1, そうでないなら 0 を返す. 略記として `!x` と書くこともできる.
- `ifthenelse(x,y,z)` ([1]94.2, 94.3.5)  
x が 0 でないなら y, そうでないなら z を返す. 略記として `x?y:z` と書くこともできる.
- `true` ([1]94.3.5)  
1 を表す.
- `false` ([1]94.3.5)  
0 を表す.
- `hex(x)` ([1]94.3.7)  
x として十進表示の整数を与えると, 十六進表示 (小文字を使用) にして返す.
- `Hex(x)` ([1]94.3.7)  
x として十進表示の整数を与えると, 十六進表示 (大文字を使用) にして返す.
- `oct(x)` ([1]94.3.7)  
x として十進表示の整数を与えると, 八進表示にして返す.
- `bin(x)` ([1]94.3.7)  
x として十進表示の整数を与えると, 二進表示にして返す.
- `min(x1,...,xn)` ([1]94.3.8)  
x1 から xn のうち最小値を返す.
- `max(x1,...,xn)` ([1]94.3.8)  
x1 から xn のうち最大値を返す.
- `veclen(x,y)` ([1]94.3.8)  
 $\sqrt{x^2 + y^2}$  を計算する.
- `array(x,y)` ([1]94.3.8)  
x は配列 (`{1,2,3,4}` のような形式で書く) で y は整数とする. このとき x の y 番目の要素を返す.



- `dim(x)` ([1]94.3.8)  
配列 `x` の要素数を返す.
- `width("文字列")` ([1]94.3.8)  
文字列 の幅を返す.
- `height("文字列")` ([1]94.3.8)  
文字列 の高さを返す.
- `depth("文字列")` ([1]94.3.8)  
文字列 の深さを返す.
- `rnd` ([1]94.3.6)  
0 ~ 1 の乱数を生成する.
- `rand` ([1]94.3.6)  
-1 ~ 1 の乱数を生成する.
- `random(x,y)` ([1]94.3.6)  
 $x \sim y$  の整数の乱数を生成する.

これらの関数と `plot` オペレーションを使うことで、簡単な関数のグラフを描くことができる.

```
\begin{tikzpicture}[domain=-0.5:4]
\draw[help lines] (-0.5,-1.1) grid (3.9,3.9);
\draw[->] (-0.5,0) -- (4.2,0) node[right] {$x$};
\draw[->] (0,-1.2) -- (0,4.2) node[above] {$y$};
\draw[color=red] plot(\x,\x) node[right] {$y=x$};
\draw[color=blue] plot(\x,{sin(\x r)}) node[right] {$y=\sin x$};
\draw[color=orange] plot(\x,{0.05*exp(\x)})
node[right] {$y=\frac{1}{20}\mathrm{e}^x$};
\end{tikzpicture}
```

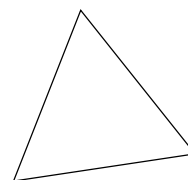


また関数は座標計算 (第 2.8 節) と組み合わせることができる. 次は指定した 2 点を頂点とする正三角形を描く例である.

```
\begin{tikzpicture}
% 2 点
\coordinate (A) at (0,0);
\coordinate (B) at (2,0.3);

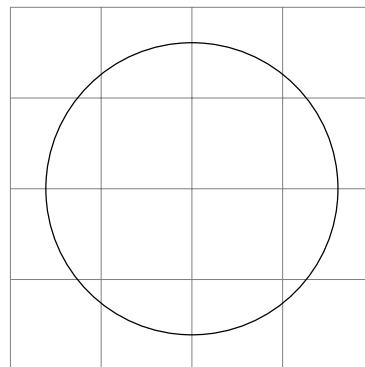
% 3 点目を計算
\coordinate (X) at ($(A)!0.5!(B)$);
\coordinate (C) at ($(X)!sin(60)*2!90:(B)$);

% 三角形
\draw (A) --(B) --(C) --cycle;
\end{tikzpicture}
```



関数を使って計算するためのコマンドとして `\pgfmathsetmacro{コマンド名}{計算式}` がある ([1]94.1.1). これは 計算式 に与えた式を計算し, 結果を コマンド名 に代入する. 次は半径  $\log(5)$  の円を描く例である.

```
\pgfmathsetmacro{\r}{\ln(5)}
\begin{tikzpicture}
\draw[help lines] (-2,-2) grid(2,2);
\draw (0,0) circle[radius=\r];
\end{tikzpicture}
```



## 2.12 レイヤーについて ([1]113)

TikZ では基本的に後に書いたパスが上に描かれるが、いわゆるレイヤーを使うことで描画の順番をコントロールすることができる。TikZ でレイヤーを使うためには、まず `\pgfdeclarelayer{レイヤー名}` というコマンドでレイヤー名を宣言しておく。すると `tikzpicture` 環境内において、`pgfonlayer` 環境によりレイヤーを設定することができる。例えば次のようにすると `layerA` と `layerB` が設定される。

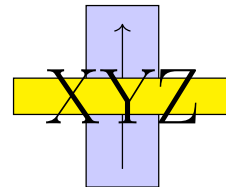
```
% レイヤー名の宣言
\pgfdeclarelayer{layerA}
\pgfdeclarelayer{layerB}

% レイヤー順の設定 (この後説明)
\pgfsetlayers{layerB,main,layerA}

\begin{tikzpicture}
% ここはレイヤーが main になる (この後説明)
\draw[fill=yellow] (-0.2, 0.8) rectangle (2.2,1.2);

\begin{pgfonlayer}{layerA} % レイヤーが layerA になる
\node[font=\Huge] at (1,1) {XYZ};
\end{pgfonlayer}

\begin{pgfonlayer}{layerB} % レイヤーが layerB になる
\draw[fill=blue!20] (0.6, 0) rectangle (1.4,2);
\draw[->] (1, 0.2) -- (1, 1.8);
\end{pgfonlayer}
\end{tikzpicture}
```

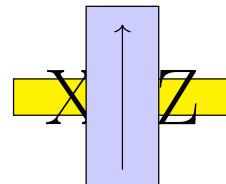


但し、デフォルトで `main` というレイヤーがあり、`pgfonlayer` 環境の外の部分は自動的に `main` レイヤーが設定された状態になる。次にレイヤーの描画順を指定するコマンドが `\pgfsetlayers{レイヤー名,...,レイヤー名}` である。このコマンドで指定したレイヤーを左から順に描画することになる。比較のためレイヤー無しの場合を載せておく。

```
% レイヤー無しの場合
\begin{tikzpicture}
\draw[fill=yellow] (-0.2, 0.8) rectangle (2.2,1.2);

\node[font=\Huge] at (1,1) {XYZ};

\draw[fill=blue!20] (0.6, 0) rectangle (1.4,2);
\draw[->] (1, 0.2) -- (1, 1.8);
\end{tikzpicture}
```



通常はレイヤーを使わなくても、上に描きたいものを後に書くようにすればよいが、レイヤーを使いたい例として次の例を挙げておく。この場合 `fit` は (f) や (g) の後に書か

ないといけないため、パスを書く順番を入れ替えることはできない。

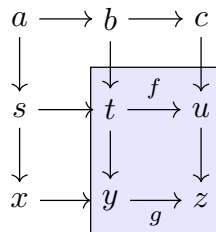
```
% この例は \usetikzlibrary{fit} が必要
\pgfdeclarelayer{background layer}
\pgfsetlayers{background layer,main}

\begin{tikzpicture}[auto]
\node (a) at (0,2){$a$}; \node (b) at (1,2){$b$}; \node (c) at (2,2){$c$};
\node (s) at (0,1){$s$}; \node (t) at (1,1){$t$}; \node (u) at (2,1){$u$};
\node (x) at (0,0){$x$}; \node (y) at (1,0){$y$}; \node (z) at (2,0){$z$};

\draw[->] (a) --(b); \draw[->] (b) --(c);
\draw[->] (s) --(t); \draw[->] (t) --node[f]{$\scriptstyle f$} (u);
\draw[->] (x) --(y); \draw[->] (y) --node[swap]{$\scriptstyle g$} (z);

\draw[->] (a) --(s); \draw[->] (s) --(x);
\draw[->] (b) --(t); \draw[->] (t) --(y);
\draw[->] (c) --(u); \draw[->] (u) --(z);

\begin{pgfonlayer}{background layer}
\node[draw,fill=blue!10,inner sep=0,fit=(t)(u)(y)(z)(f)(g)] {};
\end{pgfonlayer}
\end{tikzpicture}
```



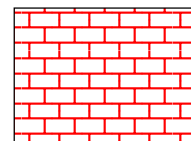
## 2.13 pattern ライブラリ ([1]15.5.1, 62)

`\usetikzlibrary{patterns}` とすることで pattern オプションを使えるようになる。これは `\path` に `[pattern=パターン名]` と設定することで、`\path` の内部を一定の「パターン」で埋めることができる。例えば次のようになる。

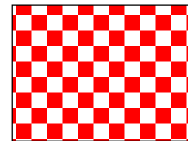
```
\tikz \draw[pattern=dots,pattern color=red]
(0,0) rectangle(2,1.5);
```



```
\tikz \draw[pattern=bricks,pattern color=red]
(0,0) rectangle(2,1.5);
```



```
\tikz \draw[pattern=checkerboard,pattern color=red]
(0,0) rectangle(2,1.5);
```

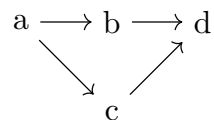


詳しくは [1]15.5.1 と 62 を参照.

## 2.14 graphs ライブラリ ([1]19)

`\usetikzlibrary{graphs}` とすることで, 有向グラフや無向グラフを簡単に描くことができる.

```
\tikz \graph { a -> {b, c} -> d };
```



詳しくは [1]19 を参照.

## 2.15 circuits ライブラリ ([1]49)

回路図を描くためのライブラリとして `circuits` ライブラリが用意されている. これを使

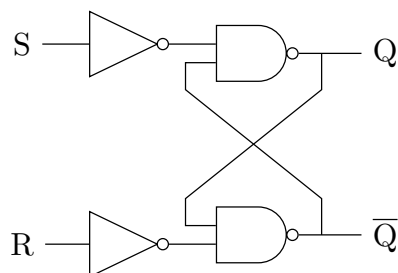
うと回路図の各要素をノードを使って描くことができる.

```
% この例は \usetikzlibrary{circuits.logic.US} が必要
\begin{tikzpicture}[circuit logic US]
% NAND ゲートと NOT ゲート
\node[nand gate] (nand1) at (2.5,2) {};
\node[nand gate] (nand2) at (2.5,0) {};
\node[not gate] (not1) at (nand1.input 1 -| 1,0) {};
\node[not gate] (not2) at (nand2.input 2 -| 1,0) {};

% 文字
\node (S) at (not1 -| 0,0) {S};
\node (R) at (not2 -| 0,0) {R};
\node (Q1) at (nand1.output -| 4,0) {Q};
\node (Q2) at (nand2.output -| 4,0) {$\overline{\mathrm{Q}}$};

% 線を引く用の座標
\coordinate (x1) at (1.8, 1.6);
\coordinate (x2) at (3.3, 0.4);

% 線を引く
\draw (S) --(not1.input);
\draw (R) --(not2.input);
\draw (not1.output) --(nand1.input 1);
\draw (not2.output) --(nand2.input 2);
\draw (nand1.output) --(Q1);
\draw (nand2.output) --(Q2);
\draw (nand1.input 2) -|(x1) --(x2) --(x2 |- nand2.output);
\draw (nand2.input 1) -|(x1 |- x2) --(x1 |- x2) --(x2 |- nand1.output);
\end{tikzpicture}
```

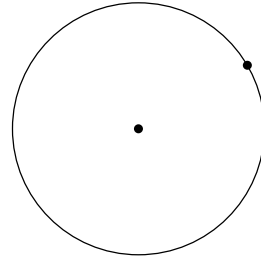


詳しくは [1]49 を参照. また似たようなパッケージとして CircuiTikZ パッケージというものもある ([5]).

## 2.16 through ライブラリ ([1]75)

`\usetikzlibrary{through}` とすることで、指定した点を通る円を描くことができる。次の例を参照。

```
\begin{tikzpicture}[point/.style={fill,
    shape=circle,inner sep=1pt,outer sep=0pt}]
\node[point] (c) at (0,0) {};
\node[point] (x) at (1.2,0.7) {};
\node[draw,circle through=(x)] at (c) {};
\end{tikzpicture}
```



## 2.17 beamer について

$\text{\TeX}$  でスライドを作る際にはよく beamer が使われているが、実は beamer と TikZ は同じ人が開発に関わっており、混ぜて使用することが簡単にできる。ここでは beamer の基本的な使い方は既知とする。

まず beamer における基本的な命令として `\pause` がある。`\pause` は `tikzpicture` 環境の中にもそのまま書くことができる。例えば次のように書いた場合、3 ページに渡って次のように表示される。

```
\begin{tikzpicture}[auto,->]
\node (a) at (0,1.2) {$a$};
\node (b) at (0,0) {$b$};
\node (x) at (1.2,1.2) {$x$};
\node (y) at (1.2,0) {$y$};

\draw (a) -- node[swap]{$\scriptstyle i$} (b);
\draw (a) -- node{$\scriptstyle f$} (x);
\pause

\draw (x) -- node{$\scriptstyle p$} (y);
\pause

\draw (b) -- node[swap]{$\scriptstyle g$} (y);
\end{tikzpicture}
```

$$\begin{array}{ccc} a & \xrightarrow{f} & x \\ i \downarrow & & \\ b & & y \end{array}$$

(1 ページ目)

$$\begin{array}{ccc} a & \xrightarrow{f} & x \\ i \downarrow & & \downarrow p \\ b & & y \end{array}$$

(2 ページ目)

$$\begin{array}{ccc} a & \xrightarrow{f} & x \\ i \downarrow & & \downarrow p \\ b & \xrightarrow{g} & y \end{array}$$

(3 ページ目)

他にも `\visible`, `\invisible`, `\uncover`, `\only`, `\onslide`, `\alt`, `\temporal` など  
も `\pause` と同様で使うことができる (省略). またこれらの命令においては `<2-3>`  
のような表記 (overlay specification というらしい. ここではオーバーレイ指定と書く) に  
よって対象となるスライドを指定することができる. このオーバーレイ指定は `node` や  
`\path` に対しても使うことができ, 例えば次のようになる.

```
\begin{tikzpicture}[auto,->]
```

```
\node (a) at (0,1.2) {$a$};
```

```
\node (b) at (0,0) {$b$};
```

```
\node (x) at (1.2,1.2) {$x$};
```

```
\node<1-2> (y) at (1.2,0) {$y$};
```

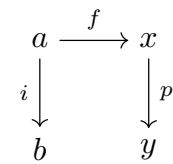
```
\draw (a) -- node[swap]{$\scriptstyle i$} (b);
```

```
\draw (a) -- node{$\scriptstyle f$} (x);
```

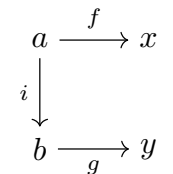
```
\draw<1> (x) -- node{$\scriptstyle p$} (y);
```

```
\draw<2-3> (b) -- node[swap]{$\scriptstyle g$} (y);
```

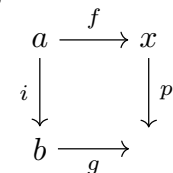
```
\end{tikzpicture}
```



(1 ページ目)



(2 ページ目)



(3 ページ目)

更に TikZ では, `\visible` 等と同様の機能を `node` や `\path` のオプションで使うこ  
ともできる (使うするには `\usetikzlibrary{overlay-beamer-styles}` が必要 ([6])).

- `[visible on=<オーバーレイ指定>]`

対象の `node` や `\path` を `<オーバーレイ指定>` の分だけ表示させる. ここでの `<オーバーレイ指定>` とは, `\visible` 等で使う記法と同じ方法で書く. 例えば



[visible on=<2-4>] のように指定する.

```

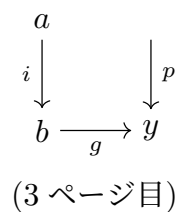
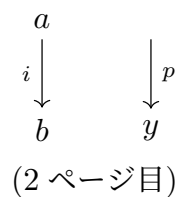
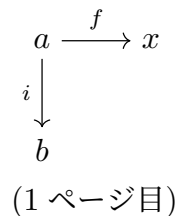
\begin{tikzpicture}[auto,->]
\node (a) at (0,1.2) {$a$};
\node (b) at (0,0) {$b$};
\node[visible on=<1>] (x) at (1.2,1.2) {$x$};
\node[visible on=<2-3>] (y) at (1.2,0) {$y$};

\draw (a) -- node[swap]{$\scriptstyle i$} (b);
\draw[visible on=<1>] (a)
    -- node{$\scriptstyle f$} (x);
\pause

\draw (x) -- node{$\scriptstyle p$} (y);
\pause

\draw (b) -- node[swap]{$\scriptstyle g$} (y);
\end{tikzpicture}

```



- [alt=<オーバーレイ指定>{設定<sub>1</sub>}{設定<sub>2</sub>}]

対象の node や \path のオプションを <オーバーレイ指定> によって切り替えることができる. <オーバーレイ指定> で指定された範囲では 設定<sub>1</sub> が適用され, そ

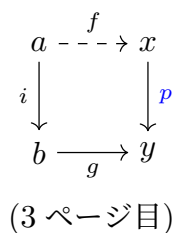
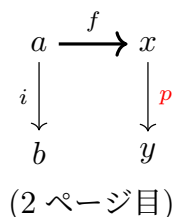
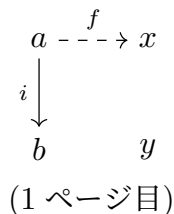
れ以外の範囲では 設定<sub>2</sub> が適用される.

```
\begin{tikzpicture}[auto,->]
\node (a) at (0,1.2) {$a$};
\node (b) at (0,0) {$b$};
\node (x) at (1.2,1.2) {$x$};
\node (y) at (1.2,0) {$y$};
```

```
\draw (a) -- node[swap]{$\scriptstyle i$} (b);
\draw[alt=<2>{line width=1pt}{dashed}] (a)
    -- node{$\scriptstyle f$} (x);
\pause
```

```
\draw (x) -- node[alt=<2>{red}{blue}]
    {$\scriptstyle p$} (y);
\pause
```

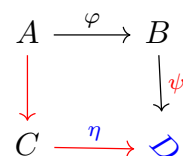
```
\draw (b) -- node[swap]{$\scriptstyle g$} (y);
\end{tikzpicture}
```



## 2.18 tikz-cd について

TikZ には, tikz-cd というパッケージが存在する. これを使用することで, 四角い図式などは行列の記法を使った記述ができるようになる. 詳しくは [7] を参照.

```
\begin{tikzcd}
A \arrow[r, "\phi"] \arrow[d, red]
& B \arrow[d, "\psi" red] \\
C \arrow[r, red, "\eta" blue]
& D
\end{tikzcd}
```

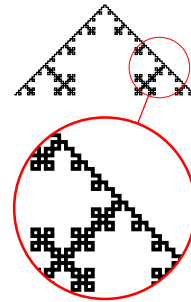


(個人的には, TikZ を使えば全部出来るので, わざわざ tikz-cd の記法を覚える必要は無いと思う.)

## 2.19 Magnifying Parts of Pictures ([1]72)

`\usetikzlibrary{spy}` とすると, `\spy` で図の一部を拡大して表示することができる.

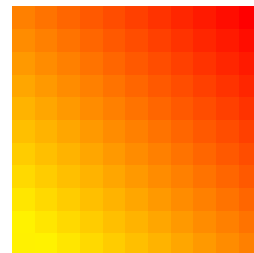
```
% この例は \usetikzlibrary{decorations.fractals} も必要
\begin{tikzpicture}
  [spy using outlines={circle,
    magnification=3,size=2cm,connect spies}]
\draw [decoration=Koch curve type 1]
  decorate { decorate{ decorate{
    decorate{ (0,0) --(2,0) }}}};
\spy[red] on (1.6,0.3) in node[left] at (2,-1.25);
\end{tikzpicture}
```



## 2.20 tikzmath コマンド ([1]58)

既に `\usetikzlibrary{calc}` とすることで様々な計算ができることを見たが, より一般的な計算をするためのコマンドとして `\tikzmath` が用意されている (これを使うには `\usetikzlibrary{math}` が必要).

```
\tikzmath{
  % 計算するための関数
  function test(\i,\j) {
    return (\i+\j)*5;
  };
  % \a{\i,\j}を計算しておく
  int \i, \j;
  for \i in {0,...,10} {
    for \j in {0,...,10} {
      \a{\i,\j} = test(\i,\j);
    };
  };
}
% \a{\i,\j}を使って絵を描く
\begin{tikzpicture}[x=0.25cm,y=0.25cm]
\foreach \i in {0,...,10} {
  \foreach \j in {0,...,10} {
    \fill[red!\a{\i,\j}!yellow]
      (\i,\j) rectangle ++(1, 1);
  }
}
\end{tikzpicture}
```



詳しくは [1]58 を参照.


## 2.21 remember picture オプション ([1]17.13.1)

まず tikzpicture 環境に [remember picture] を設定して

と書くところのように

```
\begin{tikzpicture}[remember picture,  
  baseline=(node1.base)]  
\node[draw,fill=red!20] (node1) {A};  
\end{tikzpicture}%
```


本文中の赤い四角の中に A と描かれる。

と書くところのように  本文中の赤い四角の中に A と描かれる。次に、同じように

と書くと

```
\begin{tikzpicture}[remember picture,  
  baseline=(node2.base)]  
\node[draw,fill=blue!20] (node2) {X};  
\end{tikzpicture}%
```

同様に青い四角が描かれる。

と書くと  同様に青い四角が描かれる。

この [remember picture] 有りで作ったノードの名前は別の tikzpicture 環境で参照することができる (このとき [overlay] が必要)。例えば次のように

```
\begin{tikzpicture}[remember picture,overlay]  
\draw[->,thick,green!60!black] (node1) --(node2);  
\end{tikzpicture}
```

と書くと、上の四角についての間に矢印が描かれる。

また [remember picture] と [overlay] を使用すると、現在のページ全体をノードとして (current page) によって参照することができる (形は rectangle)。これにより現在のページに対して絶対的な配置をすることができる。例えば

```
\begin{tikzpicture}[remember picture,overlay]  
\node[draw,fill=green!10,inner sep=10pt,yshift=4cm]  
  at (current page.south)  
  {『Ti\emph{k}Z の使い方』は以上です。いかがでしたか?};  
\end{tikzpicture}
```

のように書くことで現在のページの下部中央に表示することができる。

『TikZ の使い方』は以上です。いかがでしたか?

# 参考文献

- [1] Till Tantau, The TikZ & PGF Packages Manual for Version 3.1.10, January 15, 2023, <https://ctan.org/pkg/pgf>
- [2] Dr. Uwe Kern, Extending L<sup>A</sup>T<sub>E</sub>X's color facilities: the xcolor package, v3.01 (2023/11/15), <https://ctan.org/pkg/xcolor>
- [3] alg-d, TikZ で矢印を描く時の注意 (arrows.meta), <https://www.youtube.com/watch?v=5fT7ZCTIs9g>
- [4] alg-d, 『日本語 LaTeX の新常識 2021』の紹介, <https://www.youtube.com/watch?v=BXxSaG3yH6c>
- [5] CircuiTikZ, <https://ctan.org/pkg/circuitikz>
- [6] aobs-tikz, <https://www.ctan.org/pkg/aobs-tikz>
- [7] tikz-cd, <https://ctan.org/pkg/tikz-cd>

# 索引

## ■ 記号

$(x, y, z)$ , 9

$(x, y)$ , 9

(角度: 長さ), 9

| (arrow tip), 40

{ }, 20, 44

(座標<sub>1</sub>|-座標<sub>2</sub>), 76

(座標<sub>2</sub>-|座標<sub>1</sub>), 76

!, 77

", 76

\$, 77

--(座標), 10

-|(座標), 11

|- (座標), 11

$x*y$ , 85

$x+y$ , 84

$-x$ , 85

$!x$ , 88

$x-y$ , 85

$x!$ , 85

$x!=y$ , 87

$x?y:z$ , 88

$x/y$ , 85

$x\&\&y$ , 88

$x^y$ , 85

$x\leq y$ , 87

$x<y$ , 87

$x==y$ , 87

$x>=y$ , 87

$x>y$ , 87

$x||y$ , 88

['], 38

+, 59

++, 59

< (arrow tip), 40

> (arrow tip), 40

## ■ A

[above], 37

[above left], 37

[above right], 37

abs(x), 85

[absolute], 74

acos(x), 87

add(x,y), 84

[align], 40

[allow upside down], 39

[alt], 97

[anchor], 36

anchor, 32

and(x,y), 88

- angle, 33
- angles ライブラリ, 70
- /.append style, 44
- arc, 65
- Arc Barb, 41
- arc オペレーション, 65
- array(x,y), 88
- [arrows], 40
- arrows.meta ライブラリ, 6, 40
- asin(x), 87
- [at], 30
- [at end], 38
- [at start], 38
- atan(x), 87
- atan2(y,x), 87
- [auto], 38, 47

## ■ B

- [ball color], 27
- barycentric, 10
- base, 32
- base east, 32
- base west, 32
- [baseline], 41
- beamer, 95
- [behind path], 36
- [below], 37
- [below left], 37
- [below right], 37
- [bend], 66
- [bend angle], 63
- [bend at end], 66
- [bend left], 62

- [bend right], 63
- bevel, 16
- bin(x), 88
- [blend group], 46
- [bottom color], 26
- bounding box, 57
- brace, 68
- Bracket, 41
- butt, 16

## ■ C

- calc ライブラリ, 72, 77, 80
- canvas, 9
- canvas polar, 9
- ceil(x), 86
- center ([align]), 40
- center (anchor), 32
- [centered], 37
- child, 72
- child オペレーション, 72
- circle (オペレーション), 12
- circle ([shape]), 29
- Circle (arrow tip), 41
- circle オペレーション, 12
- CircuiTikZ, 94
- circuits ライブラリ, 93
- Classical TikZ Rightarrow, 41
- [clip], 27
- cmy, 55
- cmyk, 55
- [color], 19
- \colorlet , 54
- Computer Modern Rightarrow, 41

controls, 61  
`\coordinate` , 34  
`coordinate ([shape])`, 34  
`coordinate` オペレーション, 34  
`cos(x)`, 86  
`cosec(x)`, 86  
`cosh(x)`, 87  
`cot(x)`, 87  
crosses, 68  
(current bounding box), 57  
Curve-To オペレーション, 61  
`curveto`, 68  
cycle, 14

## ■ D

[dash], 17  
[dash dot], 18  
[dash pattern], 17  
[dash phase], 17  
[dashed], 18  
decorate, 67  
[decorate], 69  
decorate オペレーション, 67  
[decoration], 67  
decorations ライブラリ, 67  
decorations.fractals, 99  
decorations.pathmorphing ライブラリ,  
67  
decorations.pathreplacing ライブラリ,  
68  
decorations.shapes ライブラリ, 68  
decorations.text ライブラリ, 69  
`/.default`, 43

`\definecolor` , 55  
`deg(x)`, 86  
[densely dash dot], 18  
[densely dashed], 18  
[densely dotted], 18  
`depth("文字列")`, 89  
Diamond, 41  
`dim(x)`, 89  
`divide(x,y)`, 85  
[domain], 71  
[dotted], 18  
[double], 27  
[double distance], 27  
[draw], 15  
`\draw` , 7  
[draw opacity], 24

## ■ E

e, 85  
east, 32  
edge(座標), 65  
[edge label], 64  
[edge label'], 64  
[edge node], 64  
edge オペレーション, 65  
[end angle], 65  
`equal(x,y)`, 87  
every circle, 49  
every circle node, 49  
every edge, 50  
every label, 49  
every node, 49  
every path, 48



every picture, 49  
every pin, 49  
every pin edge, 49  
every rectangle node, 49  
every scope, 50  
every to, 49  
exp(x), 85  
extended color expression, 54

## ■ F

factorial(x), 85  
false, 88  
[fill], 18  
\fill , 19  
[fill opacity], 25  
\filldraw , 19  
[fit], 37  
fit ライブラリ, 37  
floor(x), 86  
[font], 36  
foreach, 72  
foreach オペレーション, 72  
\foreach , 72  
frac(x), 86

## ■ G

gcd(x,y), 86  
graphs ライブラリ, 93  
gray, 56  
Gray, 56  
greater(x,y), 87  
grid(座標), 13  
grid オペレーション, 13

## ■ H

height("文字列"), 89  
[help lines], 19  
hex(x), 88  
Hex(x), 88  
Hooks, 41  
hsb, 55  
Hsb, 55  
HTML, 55

## ■ I

ifthenelse(x,y,z), 88  
[in], 62  
[in front of path], 36  
[in looseness], 63  
[inner sep], 35  
[inner xsep], 35  
[inner ysep], 35  
int(x), 86  
intersections ライブラリ, 82  
iseven(x), 86  
isodd(x), 86  
isprime(x), 86

## ■ L

[label], 73  
[label distance], 74  
[label position], 74  
[left], 37  
left ([align]), 40  
[left color], 27  
less(x,y), 87  
let 式 in, 72

- let オペレーション, 72
- [line cap], 16
- [line join], 16
- Line-To オペレーション, 10
- [line width], 15
- lineto, 68
- ln(x), 85
- [local bounding box], 58
- log10(x), 85
- log2(x), 85
- [loop], 63
- [loop above], 64
- [loop below], 64
- [loop left], 64
- [loop right], 64
- [loosely dash dot], 18
- [loosely dashed], 18
- [loosely dotted], 18
- [looseness], 63

## ■ M

- math ライブラリ, 99
- max(x1,...,xn), 88
- mid, 32
- mid east, 32
- mid west, 32
- [middle color], 26
- [midway], 38
- min(x1,...,xn), 88
- [minimum height], 36
- [minimum size], 36
- [minimum width], 36
- miter, 16

- [miter limit], 16
- mod(x,y), 85
- Mod(x,y), 85
- Move-To オペレーション, 10
- moveto, 68
- multiply(x,y), 85

## ■ N

- [name path], 82
- [name prefix], 45
- [name suffix], 45
- [near end], 38
- [near start], 38
- [nearly opaque], 25
- [nearly transparent], 25
- neg(x), 85
- node{文字}, 28
- node (座標系名), 31
- node at (座標), 30
- [node contents], 28
- node (名前), 31
- \node , 30
- node オペレーション, 28
- north, 32
- north east, 32
- north west, 32
- not(x), 88
- notequal(x,y), 87
- notgreater(x,y), 87
- notless(x,y), 87
- \n 数レジスター , 73

## ■ O

`oct(x)`, 88  
`[opacity]`, 25  
`[opaque]`, 26  
`or(x,y)`, 88  
`[out]`, 62  
`[out looseness]`, 63  
`[outer sep]`, 35  
`[outer xsep]`, 36  
`[outer ysep]`, 36  
`[overlay]`, 100  
`overlay-beamer-styles` ライブラリ, 96

## ■ P

`parabola bend(座標1)(座標2)`, 66  
`parabola` オペレーション, 66  
`\path` , 6  
`[pattern]`, 92  
`patterns` ライブラリ, 92  
`perpendicular`, 76  
`\pgfmathsetmacro` , 90  
`pi`, 85  
`pic`, 70  
`/.pic`, 70  
`pic` オペレーション, 70  
`[pin]`, 75  
`[pin distance]`, 75  
`pin edge`, 75  
`pin position`, 75  
`plot coordinates{複数座標}`, 71  
`plot file{ファイル名}`, 71  
`plot(座標の計算式)`, 71  
`plot` オペレーション, 71  
`[pos]`, 30, 47

`post`, 69  
`post length`, 69  
`[postaction]`, 24, 47  
`pow(x)`, 85  
`pre`, 69  
`pre length`, 69  
`[preaction]`, 24, 47, 50  
`/.prefix style`, 44  
`\providecolor` , 56  
`\p 点レジスター` , 72

## ■ Q

`quotes` ライブラリ, 76

## ■ R

`r`, 86  
`rad(x)`, 86  
`[radius] (arc)`, 65  
`[radius] (circle)`, 12  
`raise`, 69  
`rand`, 89  
`random(x,y)`, 89  
`rect`, 16  
`rectangle(座標)`, 11  
`rectangle ([shape])`, 29  
`rectangle` オペレーション, 11  
`[relative]`, 62  
`[remember picture]`, 100  
`rgb`, 55  
`RGB`, 55  
`[right]`, 37  
`right ([align])`, 40  
`[right color]`, 27

rnd, 89  
[rotate], 21  
[rotate around], 21  
round(x), 86  
round([line cap]), 16  
round([line join]), 16  
[rounded corners], 20

## ■ S

saw, 68  
[scale], 21  
[scale around], 22  
scope 環境, 44  
scopes ライブラリ, 44  
sec(x), 86  
[semithick], 16  
[semitransparent], 25  
[shade], 26  
[shading], 26  
[shading angle], 27  
shadows ライブラリ, 24  
[shape], 29  
[sharp corners], 20  
[shift], 20  
[shift only], 22  
sign(x), 85  
sin(x), 86  
sine オペレーション, 67  
sinh(x), 87  
[sloped], 38  
snake, 68  
[solid], 18  
south, 32

south east, 32  
south west, 32  
\spy , 99  
spy ライブラリ, 99  
sqrt(x), 85  
standard color expression, 53  
[start angle], 65  
[step], 13  
Straight Barb, 41  
/.style, 43  
subtract(x,y), 85  
[swap], 38

## ■ T

tan(x), 86  
tangent, 80  
tanh(x), 87  
Tee Barb, 41  
[text], 36  
text (anchor), 32  
text along path, 69  
[text opacity], 39  
[text width], 40  
[thick], 16  
[thin], 16  
through ライブラリ, 95  
TikZ, 6  
\tikz , 8  
tikz-cd, 98  
\tikzmath , 99  
tikzpicture 環境, 6  
\tikzset , 43  
to(座標), 62

[to path], 64  
[top color], 26  
to オペレーション, 62  
[transform canvas], 23, 50  
[transform shape], 47  
[transparency group], 45  
trees ライブラリ, 72  
triangles, 69  
true, 88

## ■ U

[ultra nearly opaque], 25  
[ultra nearly transparent], 25  
[ultra thick], 16  
[ultra thin], 15  
[use as bounding box], 57  
\useasboundingbox , 57

## ■ V

veclen(x,y), 88  
[very near end], 38  
[very near start], 38  
[very nearly opaque], 25  
[very nearly transparent], 25  
[very thick], 16  
[very thin], 16  
[visible on], 96

## ■ W

wave, 55  
west, 32  
width("文字列"), 89

## ■ X

[x], 23  
\x , 71  
[x radius] (arc), 65  
[x radius] (circle), 12  
[xscale], 22  
[xshift], 21  
[xslant], 22  
[xstep], 14  
xy polar, 10  
xyz, 9  
xyz polar, 10  
\x 点レジスター , 73

## ■ Y

[y], 23  
[y radius] (arc), 65  
[y radius] (circle), 12  
[yscale], 22  
[yshift], 21  
[yslant], 22  
[ystep], 14  
\y 点レジスター , 73

## ■ Z

[z], 23  
zigzag, 68

## ■ あ

オーバーレイ指定, 96  
オプション, 8  
オペレーション, 6

## ■ か

形, 29

既定値, 8  
現在位置, 10

## ■ さ

座標, 9  
座標系名, 9  
初期値, 8  
スタイル, 43  
装飾, 67

## ■ な

名前, 31  
ノード, 28

## ■ は

パス, 7