

TikZ への入門

2024 年 1 月 21 日

目次

第 I 部	TikZ の使い方	1
第 1 章	TikZ の基礎	2
1.1	作図の要素	2
1.2	パスを指定する構文	2
1.3	座標を指定する構文	3
1.4	パス上のアクション	5
1.5	図に関するパラメータ	7
1.6	ノード	8
1.7	ツリー、グラフのための構文	11
1.8	オプションパラメータのスコープ	13
1.9	スタイル	13
第 II 部	逆引きリファレンス	15
第 2 章	線・図形編	17
2.1	直線を描く	17
2.1.1	線分の連続描画	17
2.1.2	水平線と鉛直線の組み合わせで結ぶ	19
2.2	曲線を描く	19
2.2.1	角度による指定 1	20
2.2.2	角度による指定 2	21
2.2.3	ベジエ曲線	22
2.3	矢印を描く	22
2.4	線の幅を指定する	24
2.5	線種を指定する	25
2.6	長方形を描く	26
2.7	円・楕円を描く	27

2.8	円弧を描く	28
2.9	放物線を描く	29
2.10	サインカーブを描く	30
2.11	線のつなぎ目をカスタマイズする	31
2.12	交差した線を描く	33
第 3 章	座標編	35
3.1	座標の指定	35
3.2	相対的な座標指定	36
3.3	関数を使った座標指定	37
3.4	マトリックス状に配置する	38
3.5	座標平面にグリッド線を引く	38
第 4 章	色編	41
4.1	線の色を指定する	41
4.2	内部を塗りつぶす	41
4.3	使用できる色	43
4.4	シェーディング	44
4.5	内部をパターンで塗りつぶす	46
4.6	図形の一部を塗りつぶす	47
4.7	ノードの色	48
4.8	図全体に背景色をつける	48
第 5 章	ノード・ラベル編	50
5.1	ノードを配置する	50
5.2	ノードへの命名	52
5.3	オプションによるノードの位置指定	53
5.3.1	位置指定の基本	53
5.3.2	隣接させたいノードの指定	54
5.3.3	アンカーによる指定	55
5.3.4	パスの途中のノードの配置	56
5.4	ノードの形状、大きさ	57
5.4.1	ノードの形状	58
5.4.2	ノードの輪郭とテキストとの間隔	58
5.4.3	ノードの大きさをそろえる	59
5.5	アンカーの種類	60
5.6	アンカーの指定	62
5.7	オプション指定によるラベル配置	63
5.8	ラベルの複数行表示	67
5.9	線上に白抜きラベルを配置する	70
第 6 章	平行移動、回転、拡大縮小編	71
6.1	平行移動	71

6.2	回転	72
6.3	拡大縮小	73
第 7 章	作図風コマンド編	75
7.1	鉛直線と水平線の交点の座標	75
7.2	任意のパスの交点	76
7.3	位置ベクトルの和とスカラー倍を使った座標指定	79
7.4	内分点・外分点	80
7.5	指定した 2 点間の指定した距離の座標	82
7.6	垂線との交点	83
7.7	平行線を引く	84
7.8	角度記号を描く	86
7.9	与えられた中心を持ち、与えられた点を通る円	88
7.10	座標計算を連続して記述する	89
7.11	レジスタ	90
7.12	直線に接する円を描く	91
第 8 章	スタイル編	93
8.1	スタイル定義	93
8.2	引数つきスタイル	94
8.3	every 型のスタイル	95
第 9 章	数学エンジン編	97
9.1	マクロの定義	97
9.2	演算子と数学関数	100
第 10 章	制御コマンド編	102
10.1	繰り返し制御	102
第 11 章	図の配置編	105
11.1	図の描画領域を取得する	105
11.2	図全体を枠線で囲む	106
11.3	図を並べる	108
11.3.1	figure 環境を使う	108
11.3.2	マトリックス状に並べる	112
11.3.3	tikzpicture 環境に複数並べる	113

第 I 部

TikZ の使い方

第 1 章 TikZ の基礎

この章では、TikZ を使って図を作成するときの基本となる要素を解説します。この章の執筆にあたっては、Till Tantau 氏の pgfmanual^{*1} 第 11 章を参考にさせていただきました。

1.1. 作図の要素

Inkscape などのビジュアルな操作で作図するドローソフトとは違い、TikZ ではコマンドを記述することで作図を行います。その意味では TeX で、コマンドを記述することで文書を作成するのと同じように似ています。

TikZ でコマンドを記述する際の基本要素として以下が挙げられます。

1. パスを指定する構文
2. 座標を指定する構文
3. パス上のアクション
4. 図に関するパラメータ
5. ノード
6. ツリー、グラフのための構文
7. オプションパラメータのスコープ
8. スタイル

それではこれらを詳しく見ていきましょう。

1.2. パスを指定する構文

TikZ での基本的な構文は以下の形をとります。

^{*1} <https://www.ctan.org/pkg/pgf>

\コマンド名 オペレーション列;

文末にセミコロンをつけることを忘れないようにしましょう。

コマンドには具体的には `\path` などがあります。また `\path[draw]` の省略形である `\draw` などもコマンドに含めています。

オペレーションとは描画の指定を記述したものです。座標指定やオペレーションは1つのコマンドの中で連続して記述することができ、それらを1つ以上記述したものをオペレーション列と呼びます。またその結果である線や長方形などの列をパスと呼びます。^{*2}

以下に例を示します。

コマンドとオペレーション列

```

1 \begin{tikzpicture}
2   \path[draw] (0,0) -- (1cm,0) (2cm,0) -- (3cm,0);
3 \end{tikzpicture}

```

上の例でのコマンドは `\path` です。これはパスを指定するためのコマンドになります。オペレーション列は `(0,0) -- (1cm,0) (2cm,0) -- (3cm,0)` です。

オペレーション列の中の `(0,0)`、`(1cm,0)`、`(2cm,0)` および `(3cm,0)` は座標指定です。これらは文字通り座標を表します。オペレーション列の中の `--` はオペレーションであり、ここでは直線で結ぶことを指示しています。

そして出力結果にある2つの線分が今の例のパスに相当します。図形上連結していなくても（例えば線が2つに分かれていても）パスとしては連続しているとみなします。この例では線分は2つですが、パスは1つと考えます。

ところで `[draw]` は線画で描画することを指示するものであり、このように描画の仕方を指示するものをアクションと呼びます。また `\path[draw]` は短縮して `\draw` と記述することができます。本来は `\draw` は「アクション draw が指定された `\path` コマンド」と解釈すべきですが、これも簡単に `\draw` コマンドと呼び、線画を描画するコマンドと呼ぶことにします。

本書ではパスは図（の一部）のことを、オペレーション列はその図を描画するための文（コード）のことを意味する用語として区別します。また、パスは線や形状の列のことを指しますが、厳密にはそれがどのように描画されるのか、例えば線で描画するのか塗りつぶしで描画するのかまでは指しません。線で描画する、あるいは塗りつぶしで描画するなどは、アクションで指定します。

1.3. 座標を指定する構文

座標を指定するための構文がいくつかあります。

基本となるのは x 座標と y 座標の2つの数値の組を用いて記述する、いわゆるデカルト座標です。（ x 座標, y 座

^{*2} パス (path) やオペレーション (operation) は Till Tantau 氏の pgfmanual で使われている用語です。一方、オペレーション列は本書独自の用語です。同マニュアルを見ましたがオペレーション列に相当するものをどう表現しているのか見つけることができませんでした。

標) の形式で指定します。数値の間はコンマで区切ります。

また極座標を使って(偏角: 動径長さ) の形式で指定することもできます。このときはコンマではなくコロンで区切ることに注意してください。

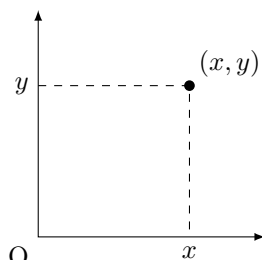


図 1.1: デカルト座標

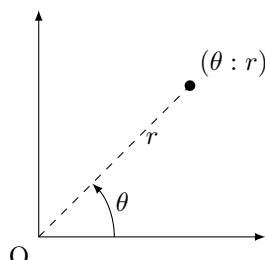


図 1.2: 極座標

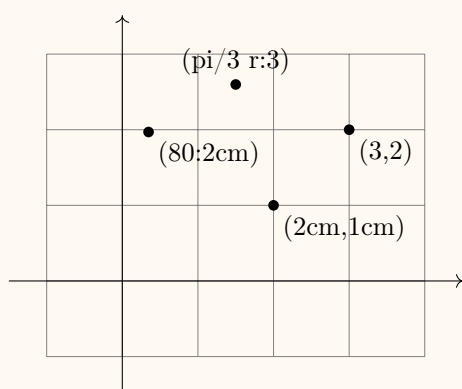
三次元の図を作成するときは、座標を(x 座標,y 座標,z 座標) の形式で指定します。

基本

```

1 \begin{tikzpicture}
2   \draw[help lines] (-1,-1) grid (4,3);
3   \draw[->] (-1.5,0) |- (4.5,0);
4   \draw[->] (0,-1.5) |- (0,3.5);
5
6   \fill (2cm,1cm) circle[radius=2pt] node[below right] {(2cm,1cm)};
7   \fill (80:2cm) circle[radius=2pt] node[below right] {(80:2cm)};
8   \fill (3,2) circle[radius=2pt] node[below right] {(3,2)};
9   \fill (pi/3 r:3) circle[radius=2pt] node[above] {(pi/3 r:3)};
10 \end{tikzpicture}

```



上の例を見てみましょう。6 行目ではデカルト座標を使って x 座標が 2cm、 y 座標が 1cm の点を指定し、そこに黒点を描画しています。7 行目では極座標を使って偏角が 80 度、動径長さが 2cm の点を指定しています。

8 行目での座標指定では cm が省略されています。座標指定において長さの単位を省略すると、既定値として cm が指定されたものとみなされます。ただし、この既定値は変更することもできます。

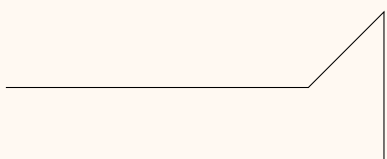
9 行目では角度の指定をラジアンで行っています。角度を表す数値の後ろに r を付加するとラジアンが単位だとみなされます。また pi は組み込み定数で円周率を意味します。

なお、使用できる長さの単位は $\text{T}_{\text{E}}\text{X}$ と同じであり、 cm や pt などが使えます。

$++$ や $+$ を使用することで、オペレーション列のカレントポジションからの相対位置で座標指定することもできます。

相対位置による座標指定

```
1 \begin{tikzpicture}
2   \draw (1,0) -- ++(4,0) -- +(1,1) -- +(1,-1);
3 \end{tikzpicture}
```



上の例を見てみましょう。最初の座標指定は $(1,0)$ と絶対座標指定で記述されています。このときカレントポジションも $(1,0)$ になります。

次の座標指定は $++$ を使って $++(4,0)$ と相対座標指定で記述されています。結果として得られる座標は $(1 + 4, 0 + 0) = (5, 0)$ となります。また $++$ を使って指定すると、カレントポジションも $(5, 0)$ に更新されます。

その次の座標指定は $+$ を使って $+(1,1)$ と相対座標指定で記述されています。結果として得られる座標は、現時点のカレントポジションが $(5,0)$ なので、 $(5 + 1, 0 + 1) = (6, 1)$ となります。 $+$ を使って指定すると、カレントポジションは更新されず、 $(5,0)$ のままです。

最後の座標指定は $+$ を使って $+(1,-1)$ と相対座標指定で記述されています。結果として得られる座標は、現時点のカレントポジションが $(5,0)$ のままなので、 $(5 + 1, 0 - 1) = (6, -1)$ となります。特に最後に描画された線分は $(6, 1)$ と $(6, -1)$ を結んでいること、つまり鉛直線になっていることが見て取れます。

$++$ と $+$ はともに相対座標指定を行います、これらの違いはカレントポジションを変更するかしないかになります。

数値ではなく他のノードからの相対位置で座標を指定することもできます。例えば **A.south** は **A** という名前のノードの下端中央の座標を表します。**south** はアンカーと呼ばれものの一つで、大きさと形状を持つノードの中のさらに細かい位置を指定するときに使います。アンカーの種類の詳細は 5.5 節を参照してください。**A.south** の座標を中心とする黒点を描画したい場合は次のように、座標を記述すべき箇所に **A.south** と記述します。

ノードのアンカーを指定

```
1 \fill (A.south) circle[radius=2pt];
```

1.4. パス上のアクション

パスとは線や形状の列のことですが、これを描画する方法として、線で描画する、塗りつぶす、シェーディングをかけるなどがあります。またクリッピング（切り抜き）などでは、描画ではなくある範囲を指定するためにパスを使うこともできます。

考え方としては、線を描くということはパスに沿ってペンを動かすことであり、塗りつぶすということは閉じたパスの中を絵具で一様に塗ることであると捉えることができます。

パスそのものを指定するコマンドは `\path` であり、

```
\path (5pt,0pt) -- (0pt,0pt) -- (0pt,5pt) -- cycle;
```

などと記述します。もちろんこれだけではパスを指定しただけですので、何も描画されません。そこで描画のアクションをオプションで指定します（オプションについては 1.5 節を参照）。

描画のアクション

```
1 \begin{tikzpicture}[fill=lightgray]
2   \path[draw] (1,0) -- (0,0) -- (0,1) -- cycle;% 線の描画
3   \path[fill, xshift=2cm] (1,0) -- (0,0) -- (0,1) -- cycle;% 塗りつぶし
4   \path[draw, fill, xshift=4cm] (1,0) -- (0,0) -- (0,1) -- cycle;% 線の描画と塗りつぶし
5 \end{tikzpicture}
```



いちばん左の例では `draw` オプションを指定することで、パスに沿って線を引きというアクションを指示しています。まんなかの例では `fill` オプションを指定することで、パスの内部を塗りつぶすというアクションを指示しています。いちばん右の例では `draw` オプションと `fill` オプションの両方を指定することで、パスに沿って線を引きつつ、内部も塗りつぶすというアクションを指示しています。

`\path[draw]` コマンドの代わりにその省略形である `\draw` コマンドを使うこともできます。さきほどの例は

描画のアクション

```
1 \begin{tikzpicture}[fill=lightgray]
2   \draw (1,0) -- (0,0) -- (0,1) -- cycle;% 線の描画
3   \fill[xshift=2cm] (1,0) -- (0,0) -- (0,1) -- cycle;% 塗りつぶし
4   \filldraw[xshift=4cm] (1,0) -- (0,0) -- (0,1) -- cycle;% 線の描画と塗りつぶし
5 \end{tikzpicture}
```



と記述することもできます。

シェーディングは `\path[shade]` や `\shade`、クリッピングは `\path[clip]` や `\clip` を使います。複数のオプションを指定した場合と同等の省略形としては例えば `\shadedraw` (`\path[shade, draw]`) があります。ただしあらゆる組み合わせに対してその省略形があるわけではありません。例えば `\drawclip` というコマンドはありません。その場合は `\draw[clip]` を使います。

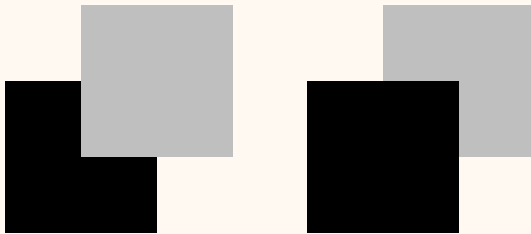
2 つの図形が互いに重なる状態で配置されたとき、どちらの図形が上位に重なるかは次のルールに従います。すなわち、後で記述した方のコマンドが上位に重なるよう描画されます。

重なる図形

```

1 \begin{tikzpicture}
2   \begin{scope}
3     \fill[black] (0,0) rectangle (2,2);
4     \fill[lightgray] (1,1) rectangle (3,3);
5   \end{scope}
6   \begin{scope}[xshift=4cm]
7     \fill[lightgray] (1,1) rectangle (3,3);
8     \fill[black] (0,0) rectangle (2,2);
9   \end{scope}
10 \end{tikzpicture}

```



左の例では、黒で塗りつぶした正方形のコードを先に、グレーで塗りつぶした正方形のコードを後に記述しています。結果としてグレーの方が黒の方の上位に重なるように描画されています。右の例はその逆ですね。

1.5. 図に関するパラメータ

線を引くあるいは塗りつぶすにあたっては、色や線の太さなど、描画の詳細な情報を指定できます。これらはオプションを指定することで行います。

TikZ ではオプションはキーと値のペアで指定します。例えば赤色を指定したい場合は `color=red` のように記述します。

オプションの指定

```

1 \begin{tikzpicture}
2   \draw[color=gray, fill=lightgray, line width=3pt] (1,0) -- (0,0) -- (0,1) -- cycle;
3 \end{tikzpicture}

```



この例では線の色 `color` を `gray`、線の幅 `line width` を `3pt` として線を描画し、塗りつぶしの色 `fill` を `lightgray` として塗りつぶしを行っています。

一部のキーは省略できます。すなわち `color` (色)、`arrows` (矢印の尖端の形状)、`shape` (ノードの形状) キーは省略することが可能です。その場合、TikZ は省略されたキーが何かを推測しますが、そのときの優先順位は `color`、`arrows`、`shape` の順となります。

キーの省略

```
1 \begin{tikzpicture}
2   \draw[gray] (1,0) -- (0,0);
3   \draw[color=gray] (1,1) -- (0,1);
4 \end{tikzpicture}
```

上の例ではいずれの線分も `gray` の色で描画されます。

1.6. ノード

TikZ では長方形、円やラベルを簡単に図に追加することができ、これらはノードを使って実現できます。`\node` コマンドを使うか、またはオペレーション列に `node` を追加すると、パス上にノードが追加されます。`\node` コマンドは `\path node` の短縮形です。

ノード

```
1 \begin{tikzpicture}
2   \node[draw] at (0,1) {\text{射}};
3   \draw[->] (0,0) node[left] {$A$} -- node[above] {$f$} (1,0) node[right] {$B$};
4 \end{tikzpicture}
```

射

$$A \xrightarrow{f} B$$

上の例の2行目では `\node` コマンドを使って、ラベルを表示するノードを配置しています。3行目では `node` を追加することで、ラベルを表示するノードを3つ配置しています。

ノードを使う場面として最もポピュラーなのはラベルの配置です。描画したいテキストを波カッコで囲みます。文字色の指定や輪郭の描画もできます。ノードの輪郭は基本的には円と長方形ですが、ライブラリを使えばより多様な形状も扱えます。波カッコの中に何も記述しなければ、つまり `{}` と記述すれば、ラベルなしの円や長方形を配置できます。

ノードの形状

```

1 \begin{tikzpicture}
2   \draw (0,0) node[draw, circle] {$A$} (1,0) node[draw, rectangle, gray] {$B$} (2,0)
      node[draw, circle, fill=gray] {};
3 \end{tikzpicture}

```



`draw` オプションは輪郭を描く、`circle` と `rectangle` はそれぞれ円、長方形ですね。色名をオプション指定すれば輪郭と文字の色を、`fill` オプションでノード内の塗りつぶし色を指定できます。

`node` の代わりに `coordinate` を指定すれば大きさがゼロのノードを配置できます。点状のノードは一見用途がなさそうですが、後述するようにノードには名前を付けることができるため、点にノード名を付けて後で参照できるようになります。

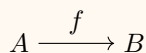
ノードの配置位置は原則としてパスのカレントポジションになります。ただし、2つのポジションの間にノードを配置することもできます。

ノード

```

1 \begin{tikzpicture}
2   \draw[->] (0,0) node[left] {$A$} -- node[above] {$f$} (1,0) node[right] {$B$};
3 \end{tikzpicture}

```



この例ではラベル A とラベル B はカレントポジションに配置されています。`(0,0) node[left] {A}` を見ると、`(0,0)` の時点でのカレントポジションが `(0,0)` ですのでその位置にラベル A が配置されます。ただしオプション `left` も指定されていますので、実際には `(0,0)` の少し左に配置されています。**(座標)** `node` のように座標の直後に `node` を記述します。あるいは `node at (座標)` のように `at` の後に座標を明示的に指定することもできます。

一方、ラベル f は2つのポジション `(0,0)` と `(1,0)` の間に配置されています。このように座標の直前（あるいはオペレーションの直後）に `node` を記述することで、直前のカレントポジションと次のカレントポジションの途中にノードを配置できます。

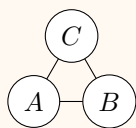
図形が重なるときは後に記述した方が上位に来ますが、ノードの場合は1つのパスが最後まで描画されたあとに配置されます。

ノードと他の図形の重なり

```

1 \begin{tikzpicture}[every node/.style={fill=white}]
2   \draw (0,0) node[draw, circle] {$A$} -- (1,0) node[draw, circle] {$B$} -- (60:1) node[draw,
      circle] {$C$} -- cycle;
3 \end{tikzpicture}

```



この例を見ると、先に3つの線分が描画されてから、その上にノードが描画されることが分かります（重なりが分かるように、すべてのノードを白で塗りつぶしています）。

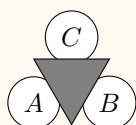
ノードが最後に描画されるというルールはあくまでも1つのパス内だけの話です。異なるパスであれば、後で記述されたコマンドの方が上位に描画されます。

ノードと他の図形の重なり2

```

1 \begin{tikzpicture}[every node/.style={fill=white}]
2   \draw (0,0) node[draw, circle] {$A$} -- (1,0) node[draw, circle] {$B$} -- (60:1) node[draw,
   circle] {$C$} -- cycle;
3   \draw[rotate around={60:(30:1/sqrt(3))}, fill=gray] (0,0) -- (1,0) -- (60:1) -- cycle;
4 \end{tikzpicture}

```



この例では、グレーで塗りつぶされた三角形を描画するコマンドが後に記述されていますので、ノードの上に重なるように描画されます。

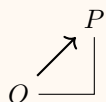
ノードには名前をつけることができます。そして後からそのノード名を指定することで、ノードを参照することができます。ノード名をつけるには `name` オプションを指定するか、`node`（ノード名）のように丸カッコでノード名を囲んで指定します。ノードを参照するには、座標を指定すべき箇所などにノード名を記述します。

ノード名と参照

```

1 \begin{tikzpicture}
2   % ノードへの命名
3   \node (O) at (0,0) {$O$};
4   \coordinate (A) at (1,0);
5   \node[name=P] at (1,1) {$P$};
6
7   % ノードの参照
8   \draw[->, thick] (O) -- (P);
9   \draw (O) -- (A) -- (P);
10 \end{tikzpicture}

```



上の例では、原点 (0,0) に配置したラベル O のノードに O という名前をつけています。同様に (1,1) に配置したラベル P のノードに P という名前をつけています。また、`\coordinate` コマンドを使用することで、座標 (1,0) に大きさのないノード、言い換えれば点状のノードを配置し、 A というノード名をつけています。

8 行目と 9 行目の `\draw` コマンドの中でノードを参照しています。本来は座標を値で指定するところにノード名を記述することで、そのノードが配置されている座標が指定されたことになります。例えば (A) と記述することで、(1,0) が指定されたのと同じ効果が得られます。

ノード名をつけるメリットの 1 つに、同じ座標値を毎回書かなくてよいことが挙げられます。実はこれは手間が省ける以上の意味を持ちます。すなわち、あとで座標値を変更する必要がでてきた場合、ノード名をつけておけばその定義箇所だけを修正すれば済みます。一方毎回座標値を直接書いていた場合は、それらすべての箇所を同じように修正しなければなりません。もし修正し忘れた箇所があると図形が壊れてしまいます。後で同じ座標値を使うことがあるのならば、ノード名をつける方が良いでしょう。

またノード名をつけるもう一つのメリットとして、コードが見やすくなるというの也有ります。特に TikZ ではなるべく見やすいコードを書き、その意味を汲み取りやすくなるようにしましょう。

1.7. ツリー、グラフのための構文

ノードはツリー構造の描画のための強力な機能も持っています。

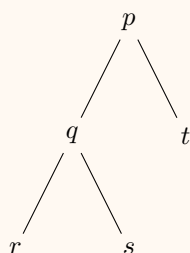
1 つの `node` の後に、`child` を続けて記述することで子ノードを追加できます。子ノードはいくつも追加できますし、子ノード自身もノードなので、子ノードの子ノードを追加することもできます。

ツリー構造

```

1 \begin{tikzpicture}
2   \node {$p$}
3     child {node {$q$}}
4       child {node {$r$}}
5       child {node {$s$}}
6     }
7   child {node {$t$}}
8   };
9 \end{tikzpicture}

```



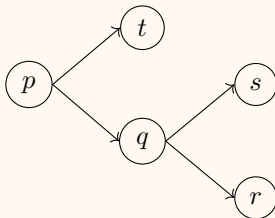
オプションを指定することで様々な描画の仕方を指示することもできます。

ツリー構造 2

```

1 \begin{tikzpicture}
2   [parent anchor=east,child anchor=west,grow=east,
3     sibling distance=15mm, level distance=15mm,
4     every node/.style={draw, circle},
5     edge from parent/.style={draw, ->}]
6   \node {$p$}
7     child {node {$q$}
8       child {node {$r$}}
9       child {node {$s$}}}
10    }
11    child {node {$t$}}
12  };
13 \end{tikzpicture}

```



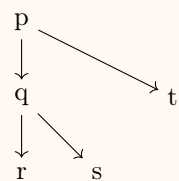
`node` コマンドを使うとノードの配置位置や見た目を細かく制御できます。しかし、グラフを描くときなど画一的なノードを配置する場合は、`graphs` ライブラリを使うことでもっと簡潔に記述することができます。

グラフ

```

1 %\usetikzlibrary{graphs}
2 \begin{tikzpicture}
3   \graph[branch right, grow down] {
4     p -> { q -> {r, s}, t }
5   };
6 \end{tikzpicture}

```



1.8. オプションパラメータのスコープ

複数のコマンドに対し同じオプションを指定したい場合があります。`scope` 環境を使うと、それらで囲まれたコマンドに対して同じオプションを設定できます。

スコープ

```

1 \begin{tikzpicture}
2   \begin{scope}[thin]
3     \draw (0,0) -- (2,0);
4     \draw (0,-.5) -- (2,-.5);
5   \end{scope}
6   \begin{scope}[ultra thick, xshift=3cm]
7     \draw (0,0) -- (2,0);
8     \draw (0,-.5) -- (2,-.5);
9   \end{scope}
10 \end{tikzpicture}

```



この例では、最初の `scope` 環境で線幅 `thin` の線分を 2 本描画しています。これらの線分を描画する `\draw` コマンド一つ一つに `thin` をオプション指定しているわけではないことに注意しましょう。にもかかわらず、描画される線分には一様に `thin` が適用されます。この `thin` が適用される範囲は、`thin` をオプション指定した `scope` 環境の中だけに限ります。

二つ目の `scope` 環境では線幅 `ultra thick` を指定し、また `xshift` を指定してスコープ内の図全体を右にずらして描画しています。

`scope` 環境は入れ子にすることができます。この場合、親のスコープと子のスコープで互いに矛盾するオプションを指定すると、子のスコープでのオプション指定が優先されます。そして個々のコマンドのオプション指定が最も優先されます。

`{tikzpicture}` 環境自身もスコープとしての働きを持ちます。この場合は、図全体に対して同じオプション指定が適用されます。

1.9. スタイル

スタイルとは特定のオプション指定を指し、オリジナルのスタイルを定義することができます。指定するオプションは複数でも構いません。一度スタイルを定義してしまえば、あとはそのスタイル名を個々の図要素のオプションとして記述するだけで、特定のオプション指定を行ったことと同等になります。

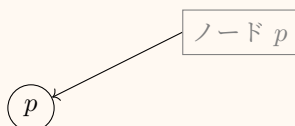
スタイル定義は、**スタイル名/.style={オプション指定}** の形式で記述します。

基本

```

1 \begin{tikzpicture}[annot/.style={gray, draw}]
2   \node[draw, circle] (p) at (0,0) {$p$};
3   \draw[<-] (p) -- (2,1) node[annot, right] {ノード $p$};
4 \end{tikzpicture}

```



この例の1行目では `gray, draw` の2つのオプション指定に `annot` という名前でスタイル定義しています。3行目では `node[annot, right]` と記述することで、ノードの描画に `annot` スタイルを適用しています。出力例を見るとグレーかつ枠線つきでラベルが描画されることが分かります。ちなみにこの例ではスタイルに `annot` と名づけましたが、これは「注釈 (annotation) を表示するためのノード」で使うことを想定してそう命名しました。

スタイルは次のような場合に使います。

1. 同じオプション指定を頻繁に使う場合
2. デザインに統一性を持たせたい場合

同じオプション指定を頻繁に使う場合は、スタイルを定義しておくことでコードの記述量を減らすことができます。例えば図中の大部分のノードに対して、「輪郭を描画し、内部をグレーで塗りつぶす」ようにしたいときに、`draw, fill=gray` というオプションを個々の `node` に指定していたら、コード量が多くなり見づらくなってしまいます。同じオプションを複数の箇所で指定するようならば、「輪郭を描画し、内部をグレーで塗りつぶす」スタイルを1つ定義しておき、そのスタイル名を `node` のオプションに指定すればコードがすっきり見やすくなります。

デザインに統一性を持たせたい場合もスタイルは有効です。スタイルを定義しておいて、そのスタイルを文書中で一貫して使うことで、統一されたデザインで文書が出力されることになります。また、あとでそのデザインを変えたい場合は、スタイル定義の部分だけを修正すれば済みます。もしスタイルを使わず、個々の箇所に直接オプション指定を行っていたならば、変更したくなった時に該当する箇所をすべて探し出す必要が生じてしまいます。

スタイル名をつけるときは、スタイルの見た目を表す名前よりも、スタイルの使われ方や意味を表す名前にした方が良いでしょう。例えば、写像を表す矢印は単線、「ならば」を表す矢印は二重線と、線種を使い分けたいとします。その場合は、スタイル名は単線、二重線をイメージさせる `single` や `double` などではなく、意味をイメージさせる `mapping` や `then` などが良いでしょう。さきほどのコード例では「注釈 (annotation) を表示するためのノード」という意味を込めて、スタイルに `annot` と命名しました。

また、たとえ同じオプション指定であっても、使われるところが違うのであれば互いに異なるスタイル名とすべきです。例えば、写像も「ならば」も同じ単線の矢印で表現するという場合であっても、スタイル名は `mapping`、`then` などと別々の名前をつけておきます。そうしておけば、あとから「ならば」だけ二重線にしたくなったら、`then` スタイルだけ変更すれば済みます。

第Ⅱ部

逆引きリファレンス

この章では、描きたいものを描くにはどのようなコマンドを使えば良いのかを、簡単に調べられるようにしてあります。TikZ は本来いろんなことができますが、ここではその中の一部を目的別に配列してあります。

第 2 章 線・図形編

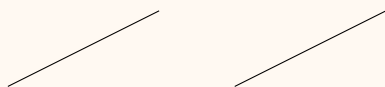
この章では、直線や曲線、円や長方形の描き方を解説します。

2.1. 直線を描く

直線（正確には線分）を描くには `\draw` コマンドを使用します。座標を 2 つ指定し、それらの間を `--` オペレーションまたは `to` オペレーションでつなぎます。

基本

```
1 \begin{tikzpicture}
2   \draw (0,0) -- (2,1); % 点 (0,0) と (2,1) の間を直線で結ぶ。
3   \draw[xshift=3cm] (0,0) to (2,1); % 点 (0,0) と (2,1) の間を直線で結ぶ。
4 \end{tikzpicture}
```



2 つのオペレーションの違いは、`to` はオプションを指定できるのに対し、`--` は指定できないことです。

2.1.1. 線分の連続描画

一つのパスで線分を連続的につなぐこともできます。一方、`--` や `to` を記述しなければ、その区間は線分が描画されません。

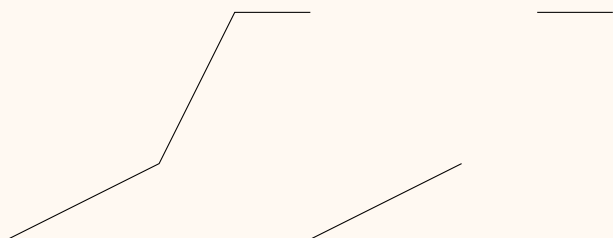
線分を連続的に描画する

```
1 \begin{tikzpicture}
2   % 点 (0,0) (2,1) 間、(2,1) (3,3) 間および (3,3) (4,3) 間を直線で結ぶ。
3   \draw (0,0) -- (2,1) -- (3,3) -- (4,3);
4 \end{tikzpicture}
```

```

5      % 点 (0,0) (2,1) 間および (3,3) (4,3) 間を直線で結ぶ。
6      \draw[xshift=4cm] (0,0) -- (2,1) (3,3) -- (4,3);
7      \end{tikzpicture}

```



線分を複数つなげて最後に閉じる場合はパスの最後に `-- cycle` を記述します。

最後に閉じる

```

1  \begin{tikzpicture}
2      \draw (0,0) -- (1,0) -- (1,1) -- cycle; %最後に (1,1) と (0,0) を結んで閉じる。
3  \end{tikzpicture}

```



上の例では3点 $(0,0)$, $(1,0)$, $(1,1)$ を頂点とする三角形を描いていますが、最後を `-- cycle` で終わらせることにより、終点 $(1,1)$ と始点 $(0,0)$ を結ぶ線分が描かれ、閉じることになります。

ところで `(0,0) -- (1,0) -- (1,1) -- cycle` と `(0,0) -- (1,0) -- (1,1) -- (0,0)` では、どちらも線が閉じますが、結果が少しだけ異なります。特に太線で描いてみると両者の違いがよく分かります。

`cycle` を使う場合と使わない場合の違い

```

1  \begin{tikzpicture}[line width=3mm]
2      \draw (0,0) -- (1,0) -- (1,1) -- cycle; %cycleで閉じる。
3      \draw[xshift=3cm] (0,0) -- (1,0) -- (1,1) -- (0,0); %始点と同じ点を指定して閉じる。
4  \end{tikzpicture}

```



上の例を見ると、左の三角形では始点で2つの線分がきれいに接続していますが、右の三角形ではそうになっていません。このように `cycle` を使うことで、TikZ はうまく接続してくれます。

同様に、2つの線分を1つのパスで連続して書くのと、別々のパスで書くのでは結果が少し異なります。

2つの線分を別々に書く場合と連続的に書く場合の違い

```

1 \begin{tikzpicture}[line width=3mm]
2   \draw (0,0) -- (1,2); \draw (1,2)-- (2,0); % 別々に書く場合
3   \draw[xshift=3cm] (0,0) -- (1,2) -- (2,0); % 連続的に書く場合
4 \end{tikzpicture}

```



左では結合部分がきれいに接続しているのに対して、右はそうになっていません。線分を連続して書くことで、TikZ は結合部分までうまく処理してくれることが分かります。

2.1.2. 水平線と鉛直線の組み合わせで結ぶ

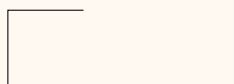
2 点間を 1 本の直線で結ぶのではなく、水平線と鉛直線の 2 本で結ぶ場合は `--` オペレーションの代わりに `|-` オペレーションまたは `-|` オペレーションを指定します。

2 点間を水平線と鉛直線の 2 本で結ぶ

```

1 \begin{tikzpicture}
2   \draw (0,0) |- (1,1); % 鉛直線ついで水平線を引く
3   \draw[xshift=2cm] (0,0) -| (1,1); % 水平線ついで鉛直線を引く
4 \end{tikzpicture}

```



オペレーション	説明
<code> -</code>	始点から鉛直線を引き、終点へ水平線を引く。
<code>- </code>	始点から水平線を引き、終点へ鉛直線を引く。

2.2. 曲線を描く

曲線を描くには `\draw` コマンドを使用します。座標を 2 つ以上指定し、それらの間を `to` オペレーションでつなぎます。このとき `to` オペレーションにオプションを指定することで多様な曲線を描画できます。

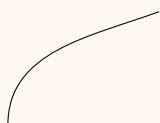
TikZ では円弧やサインカーブなどの曲線も描画できますが、ここでは一般の曲線を描く方法を述べます。

2.2.1. 角度による指定 1

1 つ目は、始点から線を出発するときの角度（以後、出射角）、および線が終点に入るとき（以後、入射角）を指定する方法です。ここで言う角度とは水平右向きを 0 度とし、反時計回りを正とします。このような曲線は一般には無限にあるわけですが、あとは TikZ がよきにはからって作図してくれます。

出射角と入射角を指定する方法

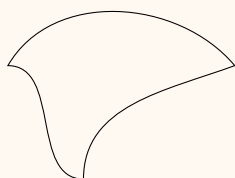
```
1 \begin{tikzpicture}
2   \draw (0,0) to[out=90,in=200] (2,1.5); % 90度方向へ出射、200度方向から入射
3 \end{tikzpicture}
```



線をつなぐための to オペレーションに、out オプションキーで出射角を、in オプションキーで入射角を指定します。上の例では曲線が始点から 90 度方向つまり鉛直上方へ伸びていき、終点へは 200 度方向から到達しています。一つのパスで曲線を連続的につなぐこともできます。

曲線を連続的に描画する

```
1 \begin{tikzpicture}
2   \draw (0,0) to[out=90,in=200] (2,1.5) to[out=130,in=60] (-1,1.5) to[out=0,in=180] cycle;
3 \end{tikzpicture}
```



出射角と入射角にどれだけの速さで近づくかを指定するには looseness オプションキーを使います。デフォルト値は 1 です。

出射角と入射角に近づく速さを指定

```
1 \begin{tikzpicture}
2   \draw (0,0) to[out=0,in=-90,looseness=0.5] (1,1);
3   \draw[xshift=1.5cm] (0,0) to[out=0,in=-90] (1,1);
4   \draw[xshift=3cm] (0,0) to[out=0,in=-90,looseness=2] (1,1);
5 \end{tikzpicture}
```



角度の起点を水平右向きではなく、始点と終点を結んだ直線の方角にしたいこともあるでしょう。その場合は

`relative` オプションキーを指定します。

角度の相対指定

```

1 \begin{tikzpicture}
2   \draw[very thick] (0,0) to[out=0,in=-90,relative] (1,1);% 角度を相対指定
3   \draw (0,0) -- (1,1);
4
5   \draw[very thick, xshift=1.5cm] (0,0) to[out=0,in=-90] (1,1);% 角度を絶対指定
6   \draw[xshift=1.5cm] (0,0) -- (1,1);
7 \end{tikzpicture}

```



左の図が `relative` を指定した例になります。始点 (0,0) から終点 (1,1) へ向かう直線、すなわち右上 45 度の方向が `out`, `in` に指定する角度の起点となります。`out=0` ですから、始点では直線と接し、`in=-90` ですから、終点では直線と直交する曲線となりました。

2.2.2. 角度による指定 2

ほかに `bend right` オプションキーと `bend left` オプションキーを使う方法もあります。

角度の相対指定 2

```

1 \begin{tikzpicture}
2   \draw[very thick] (0,0) to[bend right=60] (1,1);
3   \draw (0,0) -- (1,1);
4   \draw[very thick, xshift=1.5cm] (0,0) to[bend left=60] (1,1);
5   \draw[xshift=1.5cm] (0,0) -- (1,1);
6 \end{tikzpicture}

```



始点 (0,0) から終点 (1,1) へ向かう直線に対して、`bend right` は右に膨らんだ曲線、`bend left` は左に膨らんだ曲線になります。また、出射角は指定した相対角度、入射角は $180 - (\text{指定した相対角度})$ になります。上の例 `bend right=60` は `out=-60,in=240,relative` と同等ということになります。膨らみの量を調整するには、`looseness` オプションキーの他に `distance` オプションキーでも指定できます。

膨らみの量を指定

```

1 \begin{tikzpicture}
2   \draw[very thick] (0,0) to[bend right=60, distance=10pt] (1,1);% 10pt で指定
3   \draw[very thick, dashed] (0,0) to[bend right=60, distance=1cm] (1,1);% 1cm で指定
4   \draw (0,0) -- (1,1);

```

```
5 \end{tikzpicture}
```



2.2.3. ベジエ曲線

最後に 3 次ベジエ曲線の描き方を簡単に紹介します。なおベジエ曲線の詳細は他書に譲ります。

3 次ベジエ曲線

```
1 \begin{tikzpicture}
2   \draw (8,1) .. controls (10,2) and (10,3.5) .. (9,4); % 3 次ベジエ曲線
3 \end{tikzpicture}
```



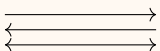
この例は、始点が (8,1)、終点が (9,4) で、制御点がそれぞれ (10,2) と (10,3.5) である 3 次ベジエ曲線になります。
and の後ろの制御点の記述を省略すると、2 つの制御点が一致したベジエ曲線になります。

2.3. 矢印を描く

`\draw` コマンドのオプションとして矢印を指定することができます。

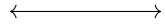

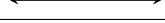



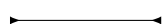






基本

```
1 \begin{tikzpicture}
2   \draw[->] (0,0) -- (2,0); % 終点に矢印の先端
3   \draw[<-] (0,-0.2) -- (2,-0.2); % 始点に矢印の先端
4   \draw[<->] (0,-0.4) -- (2,-0.4); % 始点と終点に矢印の先端
5 \end{tikzpicture}
```



終点に矢印を追加する場合は `->` を指定します。始点への追加は `<-`、両端への追加は `<->` です。

矢印の形状もさまざまなものが用意されています。具体的には次の表を見てください。`>`、`<` の部分を置き換えます。

オプション値	出力例
<code><-></code>	
<code>latex-latex</code>	
<code>stealth-stealth</code>	
<code>stealth-latex</code>	
<code><<->></code>	
<code> - </code>	
<code> -latex</code>	
<code>latex reversed-latex reversed</code>	
<code>stealth reversed-stealth reversed</code>	
<code>->>></code>	
<code>->>>></code>	
<code>->.>></code>	
<code>->>.></code>	

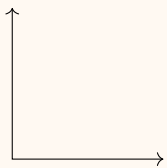
連続する線分に対して矢印を指定すると、最初の線分の始点および最後の線分の終点に矢印が追加されます。

連続する線分の矢印

```

1 \begin{tikzpicture}
2   \draw[<->] (0,2) |- (2,0);
3 \end{tikzpicture}

```



これを使うことで座標軸を描くこともできますね。

矢印のサイズをカスタマイズするときは `arrows.meta` ライブラリを使います。倍率で指定するのがおすすめです。

倍率でサイズ指定

```

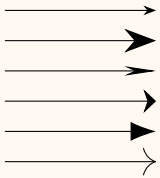
1 %\usetikzlibrary {arrows.meta}
2 \begin{tikzpicture}
3   \draw[-{Stealth}] (0,0) -- (2,0);
4   \draw[-{Stealth[scale=2.5]}] (0,-0.4) -- (2,-0.4);
5   \draw[-{Stealth[scale length=2.5]}] (0,-0.8) -- (2,-0.8);
6   \draw[-{Stealth[scale width=2.5]}] (0,-1.2) -- (2,-1.2);
7   \draw[-{Latex[scale=2]}] (0,-1.6) -- (2,-1.6);

```

```

8 \draw[->[scale=2]] (0,-2) -- (2,-2);
9 \end{tikzpicture}

```



`-{Stealth[scale=10pt]}` のように波カッコで囲むようにしてください。

2.4. 線の幅を指定する

線の幅はオプションとして指定します。スタイルとして定義されているものに、以下があります。

スタイル	線幅	出力例
ultra thin	0.1pt	
very thin	0.2pt	
thin	0.4pt	
semithick	0.6pt	
thick	0.8pt	
very thick	1.2pt	
ultra thick	1.6pt	

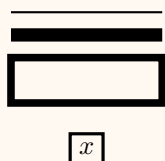
線の幅を直接指定したい場合は `line width` オプションキーを使います。線の幅のオプション指定を省略した場合は、0.4pt (`thin` スタイルに相当) がデフォルトになります。

線の太さ

```

1 \begin{tikzpicture}
2   \draw[thick] (0,0) -- (2,0); %線幅 thick (0.8pt)
3   \draw[line width=5] (0,-0.3) -- (2,-0.3); %線幅 5pt
4   \draw[line width=1mm] (0,-1.2) rectangle (2,-0.6); %線幅 1mm
5   \node[draw, very thick] at (1,-1.8) {$x$};
6 \end{tikzpicture}

```



2.5. 線種を指定する

線種（実線や破線など）はオプションとして指定します。スタイルとして定義されているものに、以下があります。

スタイル	出力例
solid	—————
dashed	- - - - -
dotted
dash dot	- . - . - .
dash dot dot	- . . - . .
loosely dashed	- - - - -
densely dashed	- - - - -
loosely dotted
densely dotted

線種のオプション指定を省略した場合は、solid スタイルがデフォルトになります。

線および間隔を細かく指定したい場合は `dash pattern` オプションキーを使います。on で線の長さを、off で空白の長さを指定します。

線と間隔の細かい指定

```

1 \begin{tikzpicture}
2   \draw[dash pattern=on 3pt off 2pt on 10pt off 5pt] (0,0) -- (5,0);
3 \end{tikzpicture}

```

上の例では、3pt の線、2pt の空白、5pt の線、3pt の空白、以下これらを繰り返した線種になります。

二重線は `double` オプションキーを指定することで描画できます。double オプションキーには二重線の間の色を指定できます。デフォルトは white になります。double distance オプションキーには二重線の間隔を指定します。double distance オプションキーを省略すると 0.6pt がデフォルトになります。

二重線のオプション

```

1 \begin{tikzpicture}
2   \draw[double distance=5pt, thick] (0,0) -- (2,0);% 線の間隔を 5pt にする。
3   \node[draw, double=red] at (0,-1) {$f(x)$};% 線の間を red にする。
4 \end{tikzpicture}

```



2.6. 長方形を描く

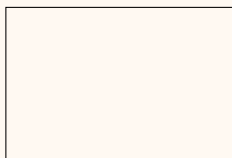
長方形を描くには `\draw` コマンド内で座標を 2 つ指定し、それらの間を `rectangle` オペレーションでつなぎます。すると、それら 2 点を対角とし、各辺が横軸、縦軸に平行な長方形が描画されます。

基本

```

1 \begin{tikzpicture}
2   \draw (0,0) rectangle (3,2);
3 \end{tikzpicture}

```



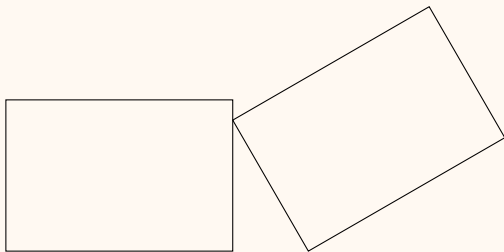
斜めに傾いた長方形を描く場合は、いったん傾いていない長方形を描き、それを回転させるという考え方で描いてください。

傾いた長方形

```

1 \begin{tikzpicture}
2   \draw (0,0) rectangle ++(3,2);% 傾ける前の長方形
3   \draw[rotate around={30:(4,0)}] (4,0) rectangle ++(3,2);% (4,0) を中心に 30 度回転した長方形
4 \end{tikzpicture}

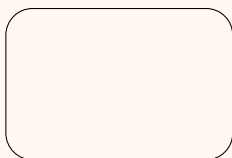
```



四隅を丸めたい場合は `rounded corners` オプションキーで四隅の円の半径を指定します。オプション値を省略すると半径 `4pt` がデフォルトです。

四隅が丸い長方形

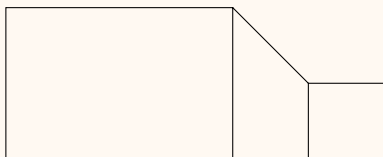
```
1 \begin{tikzpicture}
2   \draw[rounded corners=10pt] (0,0) rectangle (3,2); % 半径 10pt の円で四隅を丸める
3 \end{tikzpicture}
```



長方形についても一つのパスで連続的につなぐことができます。

パスで長方形を連続指定

```
1 \begin{tikzpicture}
2   \draw (0,0) rectangle (3,2) -- (4,1) rectangle (5,0);
3 \end{tikzpicture}
```



2.7. 円・楕円を描く

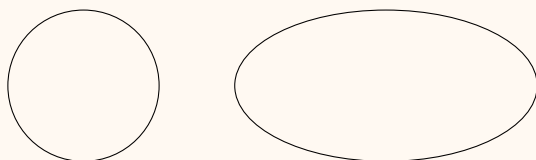
円を描くには `\draw` コマンド内で座標を1つ指定し、`circle` オペレーションで半径をオプション指定します。すると、座標指定した1点を中心とする円や楕円が描画されます。`x radius` オプションキーで水平方向の半径、`y radius` オプションキーで鉛直方向の半径を指定します。`radius` オプションキーを指定すると、`x radius` と `y radius` に同じ値が入り、結果として円が描画されることになります。

基本

```

1 \begin{tikzpicture}
2   \draw (0,0) circle[radius=1cm];
3   \draw (4,0) circle[x radius=2cm, y radius=1cm];
4 \end{tikzpicture}

```



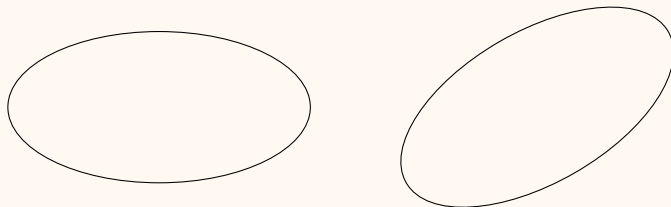
斜めに傾いた楕円を描く場合は、いったん傾いていない楕円を描き、それを回転させるという考え方で描いてください。

傾いた長方形

```

1 \begin{tikzpicture}
2   \draw (0,0) circle[x radius=2cm, y radius=1cm]; % 傾ける前の楕円
3   \draw (5,0) circle[x radius=2cm, y radius=1cm, rotate=30]; % (5,0) を中心に 30 度回転した楕円
4 \end{tikzpicture}

```



circle オペレーションの中で rotate オプションキーを指定できることに注意してください。

2.8. 円弧を描く

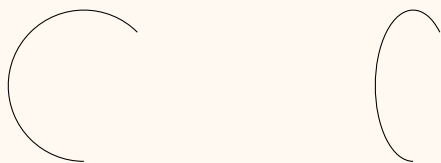
円弧を描くには \draw コマンド内で始点の座標を指定し、arc オペレーションのオプションで円弧の開始角、終了角および半径を指定します。

基本

```

1 \begin{tikzpicture}
2   \draw (0,0) arc[radius=1, start angle=45, end angle=270]; % 45 度から 270 度までの円弧
3   \draw (4,0) arc[x radius=0.5, y radius=1, start angle=45, end angle=270]; % 45 度から 270 度までの楕円弧
4 \end{tikzpicture}

```

座標として指定するのは円弧の中心ではなく、あくまでも始点であることに注意してください。上の例では点 (0,0) および (4,0) から始まる円弧がそれぞれ描画されます。

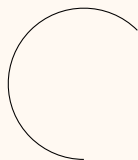
`x radius` オプションキーで水平方向の半径、`y radius` オプションキーで鉛直方向の半径を指定します。`radius` オプションキーを指定すると、`x radius` と `y radius` に同じ値が入り、結果として円の一部が描画されることになります。

`start angle` オプションキーで開始角を、`end angle` オプションキーで終了角を指定します。角度は水平右方向を 0 度とし、反時計回りが正になります。上の例ではいずれも 45 度から始まり 270 度で終わる円弧が描画されます。

終了角を指定する代わりに、`delta angle` キーオプションで開始角からの回転角を指定することもできます。

開始角とそこからの回転角を指定

```
1 \begin{tikzpicture}
2   \draw (0,0) arc[radius=1, start angle=45, delta angle=225];% 45 度から始まり 225 度進んだところで
   終わり
3 \end{tikzpicture}
```



2.9. 放物線を描く

TikZ には放物線を描くためのオペレーションがあります。数学の問題を作成するときに重宝しますね。なお、`plot` オペレーションを使っても放物線を描くことができます。「関数グラフ編」を参照してください。

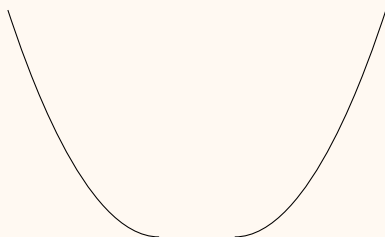
放物線を描くには `\draw` コマンドの中で `parabola` オペレーションを使うのですが、実はこのオペレーションは放物線の半分を描くものになります。`\draw` コマンド内で座標を 2 つ指定し、それらの間を `parabola` オペレーションでつなぐと、始めに指定した点を頂点とし、2 つ目に指定した点を終点とする放物線が描画されます。`bend at end` オプションを指定すると逆に、始めに指定した点を始点とし、2 つ目に指定した点を頂点とする放物線が描画されます。

基本

```

1 \begin{tikzpicture}
2   \draw (1,0) parabola (3,3);% (1,0) を頂点、(3,3) を終点とする放物線
3   \draw (-2,3) parabola[bend at end] (0,0);% (-2,3) を始点、(0,0) を頂点とする放物線
4 \end{tikzpicture}

```



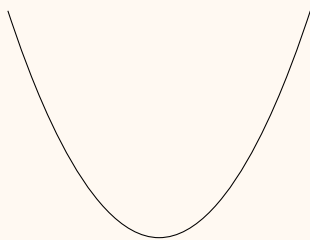
始点、頂点および終点の3点を通る放物線を描く場合は、次のように記述します。

始点、頂点および終点の3点を指定

```

1 \begin{tikzpicture}
2   \draw (-2,3) parabola bend (0,0) (2,3);% (-2,3) を始点、(0,0) を頂点、(2,3) を終点とする放物線
3 \end{tikzpicture}

```



上の例では $(-2,3)$ を始点、 $(0,0)$ を頂点、 $(2,3)$ を終点とする放物線になります。この例のように頂点の座標は自分で計算する必要があります。

2.10. サインカーブを描く

TikZ にはサインカーブを描くためのオペレーションがあります。数学の問題を作成するときに重宝しますね。なお、plot オペレーションを使ってもサインカーブを描くことができます。「関数グラフ編」を参照してください。

サインカーブを描くには `\draw` コマンドの中で `sin` および `cos` オペレーションを使うのですが、実はこのオペレーションは $[0, \pi/2]$ の区間のサインカーブを描くものになります。

`\draw` コマンド内で座標を2つ指定し、それらの間を `sin`, `cos` オペレーションでつなぎます。始めに指定した点を位相 0、2つ目に指定した点を位相 $\pi/2$ とするサインカーブが描画されます。

基本

```

1 \begin{tikzpicture}
2   \draw (0,0) sin (1,1);% (0,0)を位相 0、(1,1)を位相  $\pi/2$  とする  $\sin$  関数
3   \draw (2,1) cos (3,0);% (2,1)を位相 0、(3,0)を位相  $\pi/2$  とする  $\cos$  関数
4 \end{tikzpicture}

```



1/4 周期を超えるサインカーブを描くには、これらを組み合わせることになります。例えば、1 周期のサインカーブは次のようになります。

1 周期のサインカーブ

```

1 \begin{tikzpicture}
2   \draw (0,0) sin (pi/2,1) cos (pi,0) sin (3*pi/2,-1) cos (2*pi,0);% 1 周期の  $\sin$  関数
3 \end{tikzpicture}

```



上の例のように、一つのパスでサインカーブを連続的につなぐ記述ができることに注意してください。また `pi` は TikZ の定義済み定数であり、円周率を表します。

2.11. 線のつなぎ目をカスタマイズする

連続する線と線のつなぎ目をカスタマイズすることができます。

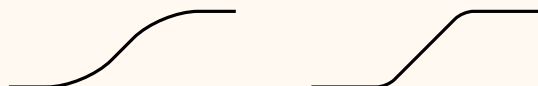
例えばつなぎ目に丸みを持たせたい場合は、`rounded corners` オプションキーを指定します。このオプションには引数として円の半径を渡すことができます。オプション値を省略すると半径 4pt がデフォルトです。

基本

```

1 \begin{tikzpicture}[very thick]
2   \draw[rounded corners=5mm] (0,0) -- (1,0) -- (2,1) -- (3,1);
3   \draw[rounded corners, xshift=4cm] (0,0) -- (1,0) -- (2,1) -- (3,1);
4 \end{tikzpicture}

```



1 番目の例では (1,0) および (2,1) の接続箇所で丸めています。

長方形のような図形でも `rounded corners` オプションキーで角に丸みを持たせることができます。

四隅が丸い長方形

```
1 \begin{tikzpicture}
2   \draw[rounded corners=10pt] (0,0) rectangle (3,2); % 半径 10pt の円で四隅を丸める
3 \end{tikzpicture}
```



パスの途中から丸みを持たせるか、あるいは持たせないかを制御することもできます。



パスの途中でつなぎ方を変える

```
1 \begin{tikzpicture}[very thick]
2   % (1,1) と (2,1) の接続部は丸くつなげる
3   \draw (0,0) -- (1,0)[rounded corners] -- (1,1) -- (2,1)[sharp corners] -- (2,0) -- (3,0);
4   \draw[xshift=4cm] (0,0) -- (1,0){[rounded corners] -- (1,1) -- (2,1)} -- (2,0) -- (3,0);
5 \end{tikzpicture}
```



1 つ目の例を見てみましょう。パスの途中から `rounded corners` オプションキーを指定することで、そこから先の接合部で丸みがつきます。逆に `sharp corners` オプションキーを指定することで、そこから先の接合部で角がつきます。2 つ目の例のように、波カッコで囲むことで `rounded corners` が効く範囲を指定することもできます。

ほかに `line join` オプションキーを指定して、接合部の形状を指定することもできます。オプション値は `round`, `bevel`, `miter` が選べます。`line join` オプションキーを省略すると `miter` がデフォルトになります。

オプション値	出力例
round	
bevel	
miter	

2.12. 交差した線を描く

ここでは線を交差させるときに、背面の線の交点の部分を消すテクニックについて解説します。

線の交差



背面の線の交点部分を消す



背面の線の交点部分を消さない

考え方は次の通りです。まず背面の線を引きます。次に前面の線を引く前に、いったん図の背景色と同じ色で前面の線と同じ線を太い線幅で引きます。最後に前面の線を引きます。

線の交差のさせ方

```

1 \begin{tikzpicture}[very thick]
2   \draw[-] (0,0) -- (1,1); % 背面の線を引く
3   \draw[-, line width=6pt, white] (0,1) -- (1,0); % 前面の線と同じ線を白い太線で引く
4   \draw[-] (0,1) -- (1,0); % 前面の線を引く
5 \end{tikzpicture}
```



背景色で引いた線の線幅の分だけ、交点にあたる部分が空くのが分かります。

ところで上のサンプルでは前面の線を2度引いています。`preaction` オプションキーを使うと1コマンドでこれを実現できます。

線の交差のさせ方2

```
1 \begin{tikzpicture}[very thick]
2   \draw[-] (0,0) -- (1,1); % 背面の線を引く
3   \draw[-, preaction={line width=6pt, white}] (0,1) -- (1,0); % 前面の線を引く
4 \end{tikzpicture}
```



`preaction` オプションキーを指定すると、引数として指定したオプションを適用した状態で描画を行い、次にそのオプションを解除した状態で再度描画を行います。上の例ではまず `line width=6pt, white` を適用して描画、次に適用を解除して描画しています。

同様に、`postaction` オプションキーは、指定したオプションをまず適用せずに描画した後、適用した状態で再度描画します。

第 3 章 座標編

この章では、座標を指定する方法を解説します。TikZ では通常の直交座標（デカルト座標）のほかに、極座標による座標指定や、関数による座標指定など多様な指定方法が用意されています。

また 2 つの曲線の交点による座標指定（第 7 章）や、繰り返し制御を併用した座標指定（第 10 章）もできます。これらはそれぞれの章を参照してください。

3.1. 座標の指定

平面図形を描くにあたって最もよく使われるのは、直交座標と極座標でしょう。直交座標は水平方向に x 軸かつ右方向を正にとり、鉛直方法に y 軸かつ上方向を正にとる座標系です。極座標は原点からの距離 r と、水平右方向から反時計回りでの角度 θ で指定する座標系です。



図 3.1: 座標の指定

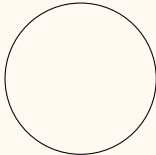
直交座標は (x,y) の書式で指定します。 $x = 2\text{cm}, y = 3\text{cm}$ ならば $(2\text{cm},3\text{cm})$ です。極座標は $(\text{theta}:r)$ の書式で指定します。 $r = 5\text{cm}, \theta = 30^\circ$ ならば $(30:5\text{cm})$ です。 θ の取りうる範囲は $-360^\circ \leq \theta \leq 720^\circ$ です。角度をラジアンで指定したい場合は $(\text{pi}/6 \text{ r}:5\text{cm})$ のように r を付加します。また、この例から分かるように、座標の指定の中に定数 (pi) （円周率）などを使用したり、式を指定することもできます。

基本

```

1 \begin{tikzpicture}
2   \draw (0,0) circle[radius=2pt];% 円の中心を直交座標で指定
3   \draw (pi/6 r:2cm + 1cm) circle[radius=1cm];% 円の中心を極座標で指定。(30:3cm)と同じ。
4 \end{tikzpicture}

```



○

3.2. 相対的な座標指定

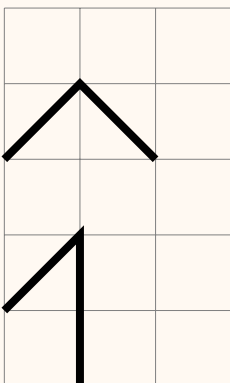
カレントポジションからの移動量を ++ や + を使って指定することで、相対的な座標指定をすることができます。

基本

```

1 \begin{tikzpicture}[line width=3pt]
2   \draw[help lines] (0,0) grid (3,5);
3   \draw (0,3) -- ++(1,1) -- ++(1,-1);% ++ を使った指定
4   \draw (0,1) -- +(1,1) -- ++(1,-1);% + を使った指定
5 \end{tikzpicture}

```



++ を使った場合は、移動後の座標が次のカレントポジションになります。上の $(0,3) \text{ -- } ++(1,1) \text{ -- } ++(1,-1)$ では $(0,3)$ から $(1,1)$ だけ移動した $(1,4)$ までを線 (--) で結びます。同時に $(1,4)$ が新たなカレントポジションになるため、そこから $(1,-1)$ だけ移動した $(2,3)$ までを線で結びます。

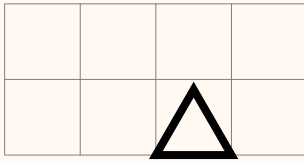
+ を使った場合は、カレントポジションは変わりません。上の $(0,1) \text{ -- } +(1,1) \text{ -- } ++(1,-1)$ では $(0,1)$ から $(1,1)$ だけ移動した $(1,2)$ までを線で結びます。しかしカレントポジションは変わらず $(0,1)$ のままなので、そこから

(1, -1) だけ移動した (1, 0) までは (1, 2) から線で結びます。

相対的な座標指定を効果的に使えば、正三角形を簡単に描くこともできます。

正三角形

```
1 \begin{tikzpicture}[line width=3pt]
2   \draw[help lines] (0,0) grid (4,2);
3   \draw (2,0) -- +(1,0) -- +(60:1) -- cycle;
4 \end{tikzpicture}
```

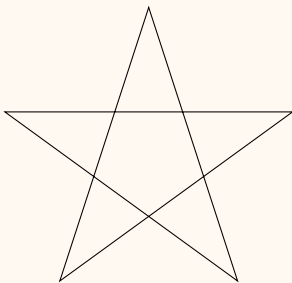


3.3. 関数を使った座標指定

座標指定するのに式や π などの定数を記述することが可能ですが、このとき \sin や \cos などの関数も使えます。

星形

```
1 \begin{tikzpicture}
2   \foreach \n in {0,1,...,4}
3     \coordinate (A\n) at ({2*cos(90+360/5*\n)}, {2*sin(90+360/5*\n)});
4   \draw (A0) -- (A2) -- (A4) -- (A1) -- (A3) -- cycle;
5 \end{tikzpicture}
```



上の例では繰り返し制御 (10.1 節参照) を併用して、 \sin や \cos を使って座標を指定しています。

式を波カッコ { } で囲むよう心がけましょう。式の形次第では波カッコがなくてもうまく評価してくれますが、一般の式の形では正しく評価されません。もしうまくコンパイルできないという時は波カッコで囲み忘れていないかチェックしてみましょう。

主な関数は 9.2 節に掲載してあります。

3.4. マトリックス状に配置する

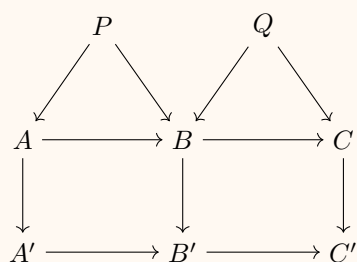
厳密に言うとこれは座標指定の構文ではないですが、複数の図をマトリックス状に配置する方法をここで解説します。

マトリックス状に配置

```

1 \begin{tikzpicture}
2   \matrix [row sep=1cm, column sep=0.5cm] {
3     & \node (P) {$P$}; & & \node (Q) {$Q$}; & \\
4     \node (A) {$A$}; & & \node (B) {$B$}; & & \node (C) {$C$}; \\
5     \node (Ap) {$A'$}; & & \node (Bp) {$B'$}; & & \node (Cp) {$C'$}; \\
6   };
7   \draw[->] (P) -- (A); \draw[->] (P) -- (B);
8   \draw[->] (Q) -- (B); \draw[->] (Q) -- (C);
9   \draw[->] (A) -- (Ap); \draw[->] (B) -- (Bp); \draw[->] (C) -- (Cp);
10  \draw[->] (A) -- (B); \draw[->] (B) -- (C);
11  \draw[->] (Ap) -- (Bp); \draw[->] (Bp) -- (Cp);
12 \end{tikzpicture}

```



上の例では `\matrix` コマンドでノードをマトリックス状に配置しています。このマトリックスは 3 行 5 列になっています。中の要素の記述の仕方は \LaTeX の `tabular` 環境と似ていますね。

`row sep` オプションキーで行と行の間の縦の間隔を指定します。ここでいう間隔とは、上の行に配置されたノードの輪郭 (5.4 節参照) の下端と、下の行に配置されたノードの輪郭の上端との間隔であることに注意してください。

同様に `column sep` オプションキーで列と列の間の横の間隔を指定します。ここでいう間隔とは、左の列に配置されたノードの輪郭の右端と、右の列に配置されたノードの輪郭の左端との間隔であることに注意してください。

3.5. 座標平面にグリッド線を引く

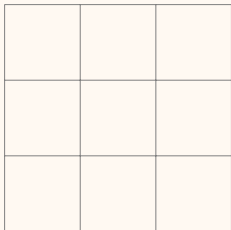
座標平面状にグリッド線を引くのもコマンド 1 つで実現できます。`\draw` コマンドのオプションに `help lines` スタイルを指定し、グリッド線を引く範囲を 2 つの座標で指定するだけです。

基本

```

1 \begin{tikzpicture}
2   \draw[help lines] (0,0) grid (3,3);% グリッド線
3 \end{tikzpicture}

```



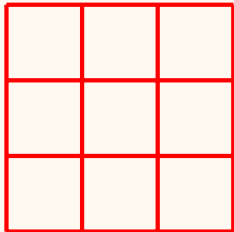
この例では (0,0) と (3,3) を対角とする長方形の中にグリッド線を描いています。help lines はスタイルの一種であり、デフォルトでは line width=0.2pt, gray でグリッド線を描画します。自分好みの線幅や色でグリッド線を描くことももちろん可能です。

線幅と色の指定

```

1 \begin{tikzpicture}
2   \draw[help lines,ultra thick, red] (0,0) grid (3,3);% 赤い太線
3 \end{tikzpicture}

```



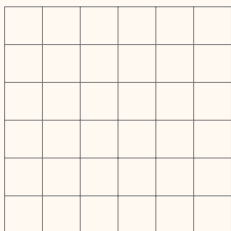
グリッドの間隔を指定するには step オプションキーを指定します。

グリッド線の間隔

```

1 \begin{tikzpicture}
2   \draw[help lines, step = 5mm] (0,0) grid (3,3);% 5mm 間隔
3 \end{tikzpicture}

```



点の位置を決める時に、目分量で決めてしまうことは多々あることと思います。簡単な図を描く際は目分量で決めて

しまうのが実際簡単ですし、点の座標を決めるのに多少の計算に基づくという場合でも、最初の数点は直接座標値を指定することになります。そこで座標を決めるための手助けとしてグリッド線を引いてから作図を始めることをお勧めします。

第 4 章 色編

この章では、線の色を指定する方法や、図形の内部を塗りつぶす方法を解説します。

4.1. 線の色を指定する

直線や曲線を描画するときの色を指定するには、`\draw` コマンドのオプションとして色名を記述します。色を指定するオプションキーは本来は `color` ですが、`color=` を省略し色名だけを記述することが可能です。

基本

```
1 \begin{tikzpicture}
2   \draw[color=red] (0,0) -- (0,1) -- (1,0);% 赤で着色
3   \draw[red, xshift=4cm] (0,0) -- (0,1) -- (1,0);% color=を省略
4 \end{tikzpicture}
```



4.2. 内部を塗りつぶす

線で囲まれた領域の内部を塗りつぶすだけならば、次のように `fill` コマンドを使い、オプションとして色を指定します。

基本

```

1 \begin{tikzpicture}
2   \fill[red] (0,0) rectangle (2,1);% 赤で塗りつぶす
3 \end{tikzpicture}

```



しかし、通常は内部を塗りつぶすだけでなく、外周となる線も描くことが多いと思います。その場合は以下のように記述します。

外周も描く

```

1 \begin{tikzpicture}[line width=6pt]
2   \draw[gray, fill=red] (0,0) rectangle +(2,1);% 赤で塗りつぶし、グレーで外周を描く。
3   \draw[fill=red, xshift=4cm] (0,0) rectangle +(2,1);% 赤で塗りつぶし、デフォルト色で外周を描く。
4 \end{tikzpicture}

```



左の例では線の色を `gray`、塗りつぶしの色を `red` としています。

`\draw` は `\path[draw]` の省略形です。したがって `\draw[gray, fill=red]` は `\path[draw=gray, fill=red]` や `\fill[red, draw=gray]` と書いても同じ結果になります。

線の色を省略するとデフォルトとして `black` が適用されます。

閉じていないパスに対して塗りつぶしをしようとすると、TikZ は始点と終点を自動的に結んだうえで塗りつぶします。

閉じていないパスに対する塗りつぶし

```

1 \begin{tikzpicture}[line width=6pt]
2   \draw[fill=red] (0,0) -- (0,1) -- (2,1) -- (2,0);% 閉じていないパス
3 \end{tikzpicture}

```



上の例では $(2,0)$ と $(0,0)$ の間が線で結ばれていませんが、塗りつぶしの結果を見ると、あたかも線で結んだうえで塗りつぶしたようになっています。一方、外周の線の方はオペレーション列として記述した通り、 $(2,0)$ と $(0,0)$ の間を結ぶ線分を描画していないことが分かります。

交差するパスでも塗りつぶしができます。

交差するパス

```

1 \begin{tikzpicture}[line width=6pt]
2   \draw[fill=red] (0,0) -- (0,1) -- (2,0) -- (2,1);% 交差するパス
3 \end{tikzpicture}

```



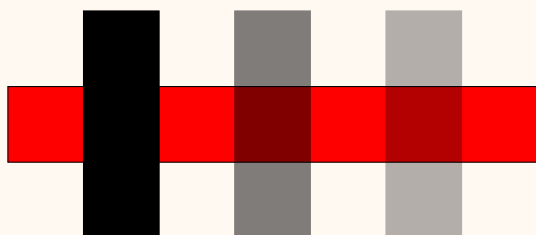
`opacity` オプションキーを指定することで透明度を指定することができます。値には 0 から 1 の間の数値を指定します。値 1 を指定すると完全に不透明、0 を指定すると完全に透明となります。

透明度の指定

```

1 \begin{tikzpicture}
2   \draw[fill=red] (0,1) rectangle +(7,1);% 背面の長方形
3   \draw[fill=black] (1,0) rectangle +(1,3);% 前面の長方形 opacity=1
4   \draw[fill=black, opacity=0.5] (3,0) rectangle +(1,3);% 前面の長方形 opacity=0.5
5   \draw[fill=black, opacity=0.3] (5,0) rectangle +(1,3);% 前面の長方形 opacity=0.3
6 \end{tikzpicture}

```



4.3. 使用できる色

TikZ は内部で `xcolor` パッケージを読み込んでいます。したがって `xcolor` と同じ方法で色の指定ができます。まず基本的な色を挙げましょう。なお最新の情報は `xcolor` パッケージの説明を参照ください。

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

またこれらの色を混ぜ合わせた色を作ることができます。

色の混合

```
1 \begin{tikzpicture}
2   \draw[fill=red!30!blue] (0,0) rectangle +(2,1);% 赤 30%、青 70%
3   \draw[fill=red!30] (0,-1.5) rectangle +(2,1);% 赤 30%、白 70%
4   \draw[fill=red!30!blue!20] (0,-3) rectangle +(2,1);% 赤 30%、青 20%、白 50%
5 \end{tikzpicture}
```



色とその割合(%)を!で区切って指定します。一つ目の例では red を 30%、blue を 70% の割合で混合しています。blue に対する割合が省略されていますが、最後の色については割合を省略することができ、合計で 100% となるよう自動で判断されます。二つ目の例では red を 30%、white を 70% の割合で混合しています。最後の色を省略すると white と判断されます。三つ目の例では 3 色を混合しており、red を 30%、blue を 20%、そして white が残りの 50% となります。

\colorlet コマンドでオリジナルの色に名前をつけることができます。

色の命名

```
1 \colorlet{MyColor}{red!30!yellow}
```

\colorlet コマンドは tikzpicture 環境外で記述します。xcolor パッケージは TikZ とは独立ですから、この色名は tikzpicture 環境外でも使用できます。

4.4. シェーディング

色で塗りつぶす際にシェーディングを行うには \shade コマンドを使います。

線形なグラデーションでシェーディングを行うには、以下のようにします。

線形なグラデーション

```

1 \begin{tikzpicture}
2   \shade[top color=red, bottom color=blue, middle color=yellow] (0,0) rectangle +(2,1);
3   \shade[draw=brown, left color=red, right color=blue, middle color=yellow] (3,0) rectangle
   +(2,1);
4 \end{tikzpicture}

```



上の左の例では上下に沿ってグラデーションをかけています。上端の色は `top color` オプションキーで指定します。例では `red` を指定しています。下端の色は `bottom color` オプションキーで指定します。例では `blue` を指定しています。中央の色は `middle color` オプションキーで指定します。例では `yellow` を指定しています。

上の右の例では左右に沿ってグラデーションをかけています。左端の色は `left color` オプションキーで指定します。例では `red` を指定しています。右端の色は `right color` オプションキーで指定します。例では `blue` を指定しています。中央の色は `middle color` オプションキーで指定します。例では `yellow` を指定しています。

右の例では `draw=brown` を指定することで、長方形の周囲の線も茶色で描画しています。`\shade` コマンドは `\path[shade]` の省略形です。ですので、`\shade[draw]` は `\path[shade, draw]`、`\draw[shade]`、`\shadedraw` と同じ意味になります。

`shading angle` オプションキーを指定すると斜めに沿ったグラデーションになります。

斜めに沿ったグラデーション

```

1 \begin{tikzpicture}
2   \shade[top color=red, bottom color=blue, middle color=yellow, shading angle=30] (0,0)
   rectangle +(2,1);
3 \end{tikzpicture}

```



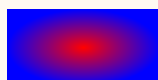
中央から放射状にグラデーションをかけるには、以下のようにします。

放射状のグラデーション

```

1 \begin{tikzpicture}
2   \shade[inner color=red, outer color=blue] (0,2) rectangle +(2,1);
3 \end{tikzpicture}

```



中央の色は `inner color` オプションキーで指定します。例では `red` を指定しています。外周の色は `outer color`

オプションキーで指定します。例では `blue` を指定しています。

球体に斜めから光を当てたようなグラデーションをかけるには、以下のようになります。

放射状のグラデーション

```
1 \begin{tikzpicture}
2   \shade[draw, ball color=red] (6,0.5) circle[radius=0.5];
3 \end{tikzpicture}
```



`ball color` オプションキーで色を指定します。上の例では `red` を指定しています。また `draw` オプションキーを指定しているので、輪郭も描画されます。今の場合 `draw` オプションキーへの色の指定を省略しているので、輪郭は黒線で描画されています。

4.5. 内部をパターンで塗りつぶす





















`patterns` ライブラリを使うと、内部を斜線などで塗りつぶすことができます。事前に `patterns` ライブラリを読みこむ必要があります。

基本

```
1 %\usetikzlibrary{patterns}
2 \begin{tikzpicture}
3   \path[pattern=north east lines, pattern color=brown] (0,0) rectangle +(2,1);
4 \end{tikzpicture}
```



`pattern` オプションキーにパターンを、`pattern color` オプションキーに色を指定します。主なパターンは以下の通りです。

パターン	見本	パターン	見本
horizontal lines		vertical lines	
north east lines		north west lines	
grid		crosshatch	
dots		crosshatch dots	
fivepointed stars		sixpointed stars	
bricks		checkerboard	
checkerboard light gray		horizontal lines light gray	
horizontal lines gray		horizontal lines dark gray	
horizontal lines light blue		horizontal lines dark blue	
crosshatch dots gray		crosshatch dots light steel blue	

4.6. 図形の一部を塗りつぶす

図形の一部を塗りつぶすテクニックとして、`scope` と `clip` を活用した例を紹介します。

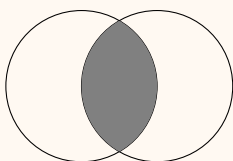
例えば集合の共通部分を示すベン図を描きたいとします。集合 A と集合 B の共通部分を塗りつぶすには以下のようになります。

集合の共通部分

```

1 \begin{tikzpicture}
2   \draw (0,0) circle [radius=1];% A
3   \draw (1,0) circle [radius=1];% B
4   \begin{scope}
5     \clip (1,0) circle [radius=1];% Bの形でクリッピングする
6     \fill[gray] (0,0) circle [radius=1];% スコープの中で A を塗りつぶす
7   \end{scope}
8 \end{tikzpicture}

```



上の例では、集合 B の形状でクリッピングを行い、その中で集合 A を塗りつぶしています。また、クリッピングが

有効なスコープを塗りつぶし処理に限定するため、`scope` 環境で `clip` コマンドと `fill` コマンドを囲んでいます。

4.7. ノードの色

ノードにも色を指定することができます。特にノード特有のオプションとしてラベルの色を指定できます。

色のみを指定（あるいは `color` オプションキーで色を指定）すると輪郭とテキストにその色が反映されます。`text` オプションキーでテキストの色を指定できます。`fill` オプションキーでノードの内部を塗りつぶすときの色を指定できます。

ノードの色

```
1 \begin{tikzpicture}
2   \node[draw, red] at (0,0) {$A$};% 前景色を指定
3   \node[draw, text=red] at (1,0) {$B$};% テキストの前景色を指定
4   \node[draw, fill=red] at (2,0) {$C$};% 塗りつぶし色を指定
5 \end{tikzpicture}
```

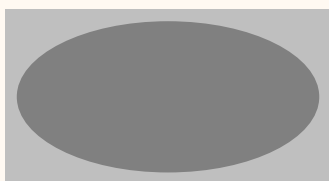


4.8. 図全体に背景色をつける

`backgrounds` ライブラリを使うことで、図全体に背景色をつけることができます。事前に `backgrounds` ライブラリを読みこむ必要があります。

基本

```
1 %\usetikzlibrary {backgrounds}
2 \begin{tikzpicture}[background rectangle/.style={fill=lightgray},
3   show background rectangle]
4   \fill[gray] (0,0) circle[x radius=2, y radius=1];
5 \end{tikzpicture}
```



`show background rectangle` スタイルを指定すると背景を描画します。このとき `background rectangle` スタイルに背景色、この例では `fill=gray`、を指定します。背景色を指定しないと枠線だけが描画されます。

その他のオプションは 11.2 節を参照してください。

第 5 章 ノード・ラベル編

この章では、ノードやラベルを作成する方法を解説します。

TikZ では図中に配置するラベル（テキスト）はノードという要素で表現します。ノードは位置やテキストのほか、色、大きさや名前などの情報も持ちます。TikZ を使いこなしていくうえで、ノードについての知識は必須です。

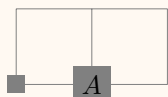
5.1. ノードを配置する

ノードを配置するには `\node` コマンドを使うのが最もシンプルです。at の後ろに座標を指定します。また波カッコ `{ }` の中にラベル（テキスト）を指定することでラベルを持つノードを描画できます。

基本

```

1 \begin{tikzpicture}
2   \draw[help lines] (0,0) grid (2,1);
3   \node[fill=gray] at (0,0) {}; % ノードのみを配置
4   \node[fill=gray] at (1,0) {$A$}; % ラベルを持つノード
5 \end{tikzpicture}
```



上の例では (0,0) にノードのみを配置し、(1,0) にラベル付きのノードを配置しています。またノードの大きさを示すために背景色を `gray` にしました。ラベルを持たない場合でも `{ }` のように波カッコを記述しなければならないことに注意してください。この例では一つのコマンドで一つのノード配置のみを行っていますが、次のように一つのパス内でノードを次々に配置することもできます。

複数のノード

```

1 \begin{tikzpicture}
2   \draw (0,0) -- (1,0) node {$A$} -- (2,0) node {$B$}; % 線分を引くと同時にラベルを配置
3   \draw (0,-1) (1,-1) node {$A'$} (2,-1) node {$B'$}; % ラベルのみを配置
4 \end{tikzpicture}

```

座標指定の後に `node` オペレーションを記述すると、その座標の位置にノードを配置します。上の例では座標 (1,0) を指定した後にラベル *A* を持つ `node` オペレーションを記述しているため、(1,0) の位置に *A* というラベルのノードが配置されることになります。直前に指定された座標ではなく、任意の座標を指定したい場合は `at` を使います。

ノード配置の方法として `\node` コマンドを使う方法と `node` オペレーションを使う方法とを述べましたが、前者は `\path node` の短縮形であり、実際は両者は同じものです。

座標指定の前に `node` オペレーションを記述することで、線分や曲線の途中にラベルを配置することもできます。

線分の途中にノードを配置

```

1 \begin{tikzpicture}
2   \draw (0,0) -- node {$A$} (1,0) -- node {$B$} (2,0);
3   \draw (0,-1) to[bend left=30] node {$A$} (1,-1) to[bend right=30] node {$B$} (2,-1);
4 \end{tikzpicture}

```

座標指定の前に `node` オペレーションを記述すると、前の座標と次の座標の中間位置にノードを配置します。少し紛らわしいですが、座標指定の前に `node` オペレーションを記述するのと、後に記述するのとでノードの配置位置が異なることに注意してください。中間位置ではなく、任意の相対位置にノードを配置するには `pos` オプションキーを使います。引数には 0 から 1 の間の数を与えます。

相対位置を指定

```

1 \begin{tikzpicture}
2   \draw (0,0) -- node[pos=0.1] {$A$} (5,0); % 左から 0.1:0.9 の位置に配置
3   \draw (0,-1) -- node[midway] {$B$} (5,-1); % 中間の位置に配置
4 \end{tikzpicture}

```

上の例のように `pos=0.1` と指定すると、始点 (0,0) から終点 (5,0) に向かって 0.1 : 0.9 の比の位置にノードが配置されます。`midway` スタイルは `pos=0.5` と同等です。スタイルには以下のものがあります。

スタイル	説明	出力例
at start	<code>pos=0</code>	A ————
very near start	<code>pos=0.125</code>	— A ————
near start	<code>pos=0.25</code>	—— A ————
midway	<code>pos=0.5</code>	——— A ———
near end	<code>pos=0.75</code>	———— A —
very near end	<code>pos=0.875</code>	————— A —
at end	<code>pos=1</code>	————— A

5.2. ノードへの命名

よく参照するノードには名前をつけると便利です。`node` コマンドまたは `node` オペレーションの直後に、(名前) のように指定することでノードに名前をつけることができます。

基本

```

1 \begin{tikzpicture}
2   \node (A) at (0,0) {$A$};% A と命名
3   \draw (2,0) -- node (B) {$B$} (4,0) -- (6,0) node (C) {$C$} ;
4 \end{tikzpicture}
```

A ——— B ————— C

上の例では、(0,0) の位置にあるノードに A 、(2,0) と (4,0) の中間にあるノードに B 、そして (6,0) の位置にあるノードに C と命名しています。

ところで、実際の作図にあたっては、ノードというより点に命名したい場合があります。そのときは大きさ 0 のノードを作成するための `\coordinate` コマンドを使います。

基本

```

1 \begin{tikzpicture}
2   \coordinate (D) at (1,-3);% 大きさ 0 のノード
3 \end{tikzpicture}
```


`\node` の場合は命名を省略することもできますが、`\coordinate` の場合は命名の省略はできません。命名したノードは後続のコマンドで参照することができます。

ノード名の参照

```

1 \begin{tikzpicture}
2   \coordinate (A) at (0,0);
3   \coordinate (B) at (1,0);
4   \draw (A) -- (B);% ノード名を参照
5   \node[below] at (A) {$A$};% ノード名を参照
6 \end{tikzpicture}

```



上の例のように、座標指定のところで数値で指定する代わりに（ノード名）と指定します。

5.3. オプションによるノードの位置指定

ここではノードを配置するときの、指定した座標からの相対位置の指定の方法を説明します。

5.3.1. 位置指定の基本

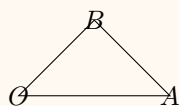
指定した座標にノードを配置したときの厳密な位置ですが、既定では次のように座標の位置とノードの中心の位置が一致するようにノードが配置されます。

既定のノード位置

```

1 \begin{tikzpicture}
2   % 座標と同じ位置にノード配置
3   \draw (0,0) node {$O$} -- (2,0) node {$A$} -- (1,1) node {$B$} -- cycle;
4 \end{tikzpicture}

```



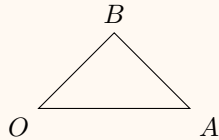
ラベル O のノードはその直前の座標 $(0,0)$ に配置されるわけですが、このときラベル O の中心が $(0,0)$ に来ていることが分かります。もし隣接する位置にラベルを配置したい場合は、`node` オペレーションに配置を示すオプションを指定します。

隣接する位置に配置

```

1 \begin{tikzpicture}
2   \draw (0,0) node[below left] {$O$} -- (2,0) node[below right] {$A$} -- (1,1) node[above]
   {$B$} -- cycle;% 隣接する位置にノードを配
   置
3 \end{tikzpicture}

```



上の例ではラベル O はその直前の座標 $(0,0)$ の左下、ラベル A はその直前の座標 $(2,0)$ の右下、そしてラベル B はその直前の座標 $(1,1)$ の上に配置されます。なおラベル O のノード自身が $(0,0)$ の左下に位置することに注意しましょう。`below left` オプションキーはノードから見たラベル（テキスト）の表示位置ではなく、ラベル O のノードそのものの配置位置を指定します。

配置に関するオプションキーには以下があります。

オプションキー	位置
<code>above</code>	上
<code>below</code>	下
<code>left</code>	左
<code>right</code>	右
<code>above left</code>	左上
<code>above right</code>	右上
<code>below left</code>	左下
<code>below right</code>	右下
<code>centered</code>	中央

5.3.2. 隣接させたいノードの指定

`node[above of=ノード名]` と指定すると隣接させたいノードを任意に指定できます。

隣接するノードを指定

```

1 \begin{tikzpicture}
2   \node (A) at (0,0) {$A$};% 起点となるノード
3   \node[left of=A] {$B$};% left of で指定
4   \node[left] at (A) {$C$};% at で指定
5 \end{tikzpicture}

```

$B \quad CA$

上の例ではいずれも A の左にノードを配置しています。ラベル B のノードでは `left of`、ラベル C のノードでは `at` で指定しています。見て分かるように隣接の距離に違いがあります。

`above`、`below`、`left` および `right` オプションキーでは隣接するときの距離（オフセット値）を指定することもできます。そのときは `left=オフセット値` のように指定します。

隣接するときの距離を指定

```
1 \begin{tikzpicture}
2   \node (A) at (0,0) {$A$};% 起点となるノード
3   \node[left=20pt] at (A) {$B$};% 20pt のオフセット
4 \end{tikzpicture}
```

$B \quad A$

隣接するノードとその距離の同時指定は `positioning` ライブラリを使うことで可能です。事前に `positioning` ライブラリを読みこむ必要があります。

`left=20pt of A` のような書式でオプション指定できます。

隣接するノードと距離を同時に指定

```
1 %\usetikzlibrary{positioning}
2 \begin{tikzpicture}
3   \node (A) at (0,0) {$A$};% 起点となるノード
4   \node[left=20pt of A] {$B$};% 20pt のオフセット
5 \end{tikzpicture}
```

$B \quad A$

5.3.3. アンカーによる指定

`above` や `left` などと同じ効果をもたらすオプションとして `anchor` があります。まずは例をご覧ください。

アンカーによる指定

```
1 \begin{tikzpicture}
2   \node (A) at (0,0) {$A$};% 起点となるノード
3   \node[anchor=south] at (A) {$B$};% 新しいノードの下のアンカーを A の位置に合わせる
4 \end{tikzpicture}
```

B
 A

コードを見ると A の下（南）にラベル B を配置するように一見思ってしまいますが、実際には A の上に配置されます。これは「これから配置するノードの下位置（south）を A の位置 (0,0) に合わせる」という指示になっているためです。アンカーとは自身のノードのどの位置を、指定した座標に合わせるかを意味します。

アンカーには主に以下があります。

アンカー	位置
south	下のアンカー
north	上のアンカー
west	左のアンカー
east	右のアンカー
south east	右下のアンカー
north west	左上のアンカー
south west	左下のアンカー
north east	右上のアンカー

これ以外にもアンカーはあります。5.5 節を参照ください。

使用例をもう少し挙げます。

アンカーによる指定 2

```

1 \begin{tikzpicture}
2   \node (A) at (0,0) {$A$}; % 起点となるノード
3   \node[anchor=south west] at (A) {$B$}; % 新しいノードの左下のアンカーを A の位置に合わせる。
4   \node[anchor=north east] at (A) {$C$}; % 新しいノードの右上のアンカーを A の位置に合わせる。
5 \end{tikzpicture}

```

上の例ではラベル B のノードの左下のアンカー位置と A の位置を合わせています。結果的に A の右上に B が配置されます。また、ラベル C のノードの右上のアンカーと A を合わせています。結果的に A の左下に B が配置されます。このように above や below と比べると直観に反する配置になるので、慣れが必要です。その代わり、アンカーには多くの種類があるため細かい制御ができる点が強みです。

5.3.4. パスの途中のノードの配置

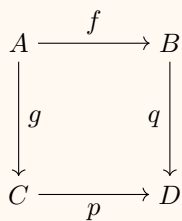
パスの途中にノードを配置するときのための、便利な位置指定のオプションがあります。特にオプションを指定しないと線上にノードが配置されてしまいますが、auto オプションを指定すると自動的に線の脇にノードが配置されるようになります。通常は tikzpicture 環境に auto オプションキーを指定します。

パスの途中のノード

```

1 \begin{tikzpicture}[auto]
2   \node (A) at (0,2) {$A$}; \node (B) at (2,2) {$B$};
3   \node (C) at (0,0) {$C$}; \node (D) at (2,0) {$D$};
4   \draw[->] (A) -- node {$f$} (B);% 右向き矢印
5   \draw[->] (A) -- node {$g$} (C);% 下向き矢印
6   \draw[->, swap] (C) -- node {$p$} (D);% 右向き矢印
7   \draw[->, swap] (B) -- node {$q$} (D);% 下向き矢印
8 \end{tikzpicture}

```



上の例では線の脇にラベルが配置されていますが、 f と g を見るとパスの進行方向に向かって左側に配置されることが分かります。逆に、進行方向に向かって右側に配置するには p と q のように `swap` オプションキーを指定します。曲線の途中にラベルを配置するのに、線に沿うようにするには `sloped` オプションキーを使います。

曲線に沿うラベル

```

1 \begin{tikzpicture}
2   \draw (0,0) to[out=0, in=270] node[above, sloped, near start] {start} node[below, sloped,
3     near end] {end} (5,2);% 曲線上のラベル
4 \end{tikzpicture}

```



5.4. ノードの形状、大きさ

ノードは形状と大きさを持ちます。例えば `node` のオプションとして `draw` オプションキーを指定すると、ノードの形状と大きさにしたがった輪郭が描画されます。

5.4.1. ノードの形状

ノードの形状を指定するには `node` に形状を表すシェイプを指定します。標準では長方形 (`rectangle`) と円 (`circle`) の2つが用意されており、デフォルトは `rectangle` です。

ノードの形状を指定し、輪郭を描画する

```

1 \begin{tikzpicture}
2   \node[draw, rectangle] at (0,0) {ノード};% 長方形
3   \node[draw, circle] at (3,0) {ノード};% 円
4   \node[draw] at (6,0) {ノード};% デフォルトでは rectangle
5 \end{tikzpicture}

```



上の例では `draw` オプションキーを `node` に指定しているので、それぞれのノードの形状に沿った輪郭が描画されています。

実は `coordinate` シェイプというものもありますが、これは `\coordinate` コマンドによるノード定義と同様、大きさのないノードになるにすぎません。`coordinate` シェイプは「大きさが無い」という形状を意味するシェイプです。

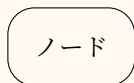
`rectangle` の場合のオプションとして `rounded corners` オプションキーを指定すると角が丸くなります。

角が丸い長方形

```

1 \begin{tikzpicture}
2   \node[draw, rounded corners=10pt, inner sep=10pt] (A) at (0,0) {ノード};
3 \end{tikzpicture}

```



5.4.2. ノードの輪郭とテキストとの間隔

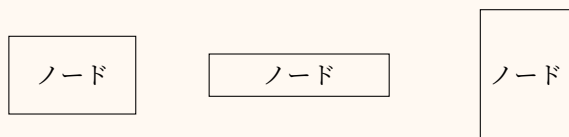
ノードの輪郭とテキストの間隔を制御することができます。`node` のオプションキーとして `inner sep`、`inner xsep` または `inner ysep` を指定します。

輪郭とテキストの間隔

```

1 \begin{tikzpicture}
2   \node[draw, inner sep=10pt] at (0,0) {ノード};% 上下左右を広げる
3   \node[draw, inner xsep=20pt] at (3,0) {ノード};% 左右を広げる
4   \node[draw, inner ysep=20pt] at (6,0) {ノード};% 上下を広げる
5 \end{tikzpicture}

```



`inner sep` オプションキーは上下左右、`inner xsep` オプションキーは左右、そして `inner ysep` オプションキーは上下の間隔を制御します。CSS で言うところの `padding` と同じ概念にあたります。

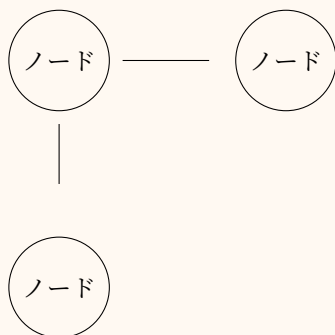
一方、輪郭とその外との間隔を制御することもできます。`node` のオプションキーとして `outer sep`、`outer xsep` または `outer ysep` を指定します。これは CSS で言うところの `margin` と同じ概念にあたります。

輪郭とその外との間隔

```

1 \begin{tikzpicture}
2   \node[draw, circle, outer sep=5pt] (A) at (0,0) {ノード};% 上下左右の間隔を空ける
3   \node[draw, circle, outer xsep=10pt] (B) at (3,0) {ノード};% 左右の間隔を空ける
4   \node[draw, circle, outer ysep=20pt] (C) at (0,-3) {ノード};% 上下の間隔を空ける
5   \draw (B) -- (A) -- (C);% 線で結んでみる
6 \end{tikzpicture}

```



このオプションの主な使いどころは、アンカーの位置を制御したいときです。上の例では3つのノードの間を線で結んでいますが、線の端点がノードの輪郭に接続していないことが見て取れます。これは線の接続先であるアンカーが輪郭の外側に移動したためです。`outer sep` オプションキーは上下左右、`outer xsep` オプションキーは左右、そして `outer ysep` オプションキーは上下の間隔を制御します。

5.4.3. ノードの大きさをそろえる

ノードの大きさは中のテキストに合わせて自動的に設定されます。ところでノードをいくつか並べた時に、それらのサイズをそろえたいということがあります。その場合はノードのサイズの最小値を設定することで、それより小さいサイズのテキストであってもノードの大きさを確保することができます。

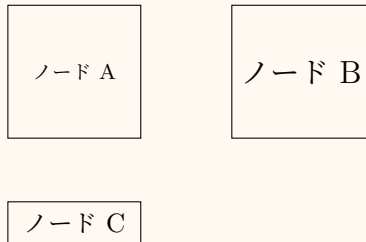
`minimum size` オプションキーはノードの幅と高さ、`minimum height` オプションキーは高さ、そして `minimum width` オプションキーは幅の最小値を指定します。

ノードのサイズの最小値

```

1 \begin{tikzpicture}
2   \node[draw, minimum size=50pt] (A) at (0,0) {\footnotesize{ノード A}};% 幅と高さの最小値
3   \node[draw, minimum height=50pt] (B) at (3,0) {\large{ノード B}};% 高さの最小値
4   \node[draw, minimum width=50pt] (C) at (0,-2) {ノード C};% 幅の左右の最小値
5 \end{tikzpicture}

```



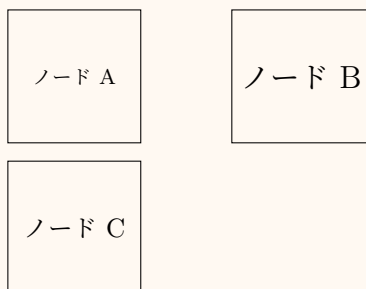
ノードのサイズをそろえるためには各ノードに同じ最小値を設定する必要があります。ですので、通常はスタイルとして設定します。

ノードの最小値 (スタイル設定)

```

1 \begin{tikzpicture}[box/.style={minimum size=50pt}]% 幅と高さの最小値をスタイル設定
2   \node[draw, box] (A) at (0,0) {\footnotesize{ノード A}};
3   \node[draw, box] (B) at (3,0) {\large{ノード B}};
4   \node[draw, box] (C) at (0,-2) {ノード C};
5 \end{tikzpicture}

```

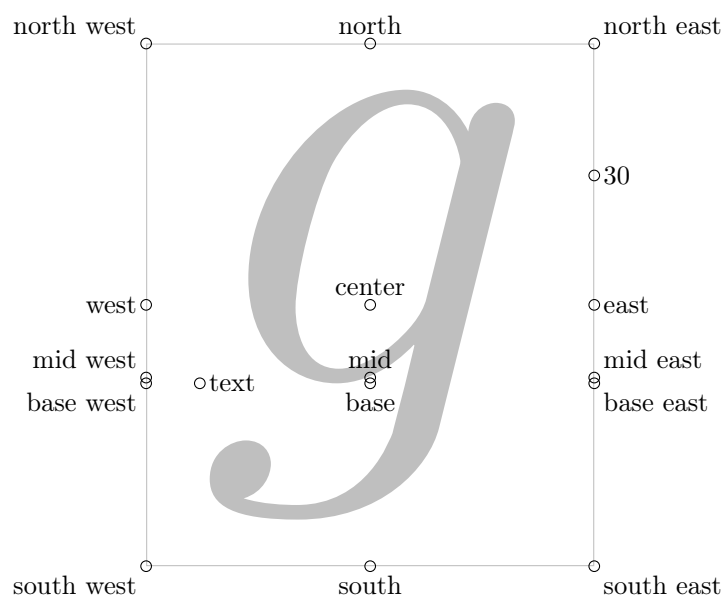


上の例ではノードのサイズの最小値を定義したスタイル `box` を定義し、各ノードで `box` のみ記述することで、共通の最小値を指定しています。

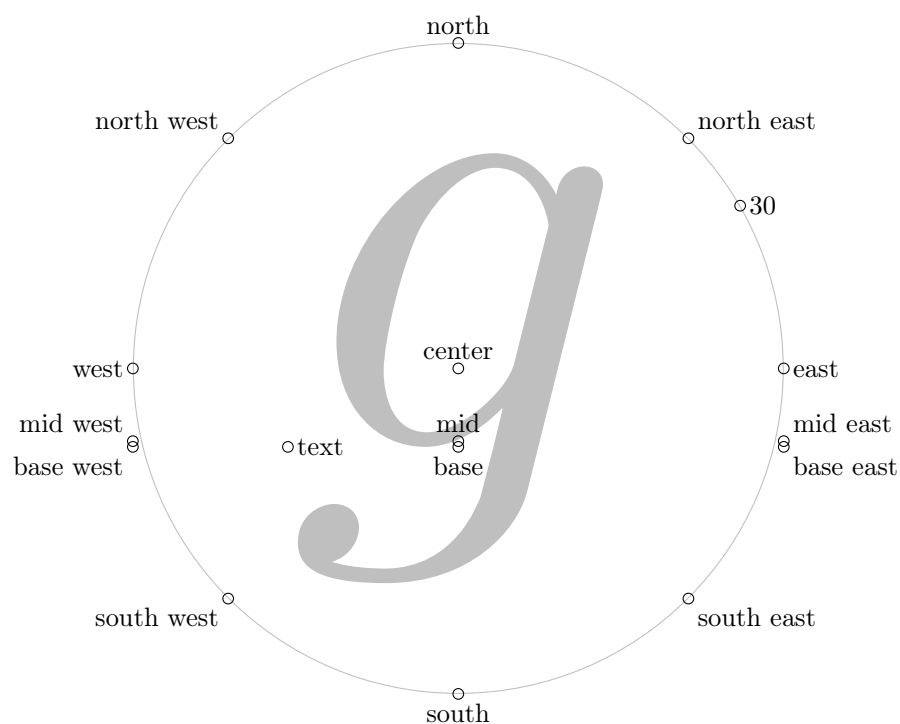
5.5. アンカーの種類

ここではアンカーとその位置を図示します。ノードの形状が `rectangle` の場合、および `circle` の場合は、それぞれ次のようになります。

長方形の場合



円の場合



上の図の 30 と書かれている位置は、center から 30 度の方向（水平右向きを 0 度とする）に引いた直線と、ノードの輪郭との交点の位置を意味します。

5.6. アンカーの指定

A という名前のノードに対して `A.center` と記述すると、5.5 節で示した図の `center` の位置を指し示すことになります。たとえば座標指定のところで (`A.center`) と記述することで、ノード A の中心位置を座標として指定することになります。

アンカーを省略するとデフォルトとして `center` が指定されたものとみなされます。例えば O という名前のノードに対して、「ノード O の位置に円を描く」(`\draw (O) circle[radius=2pt];`) とすると、次の出力例のように円は `O.center` の位置を中心に描かれます。

ノード O の位置に円を描く

```
1 \begin{tikzpicture}
2   \node[draw] (O) at (0,0) {\Huge O}; % 基準となるノード
3   \draw (O) circle[radius=2pt]; % ノード O の位置に円を描く
4 \end{tikzpicture}
```



もし `O.base` の位置を中心に円を描きたい場合は、次のように (`O.base`) を座標指定のところに記述します。

ノード O のベースの位置に円を描く

```
1 \begin{tikzpicture}
2   \node[draw] (O) at (0,0) {\Huge O}; % 基準となるノード
3   \draw (O.base) circle[radius=2pt]; % ノード O のベースの位置に円を描く
4 \end{tikzpicture}
```



一方、新しいノードを作成するときに `anchor` オプションキーを省略するとやはり `center` が指定されたものと見なされます。例えば「(0,0) の位置にノード O を配置する」(`\node (O) at (0,0) {\Huge O};`) とすると、次のように (0,0) の位置と `O.center` の位置が一致するようにノードが配置されます。

指定した位置にノードを配置する

```
1 \begin{tikzpicture}
2   \draw (0,0) circle[radius=2pt]; % 原点に目印
3   \node[draw] (O) at (0,0) {\Huge O}; % (0,0) の位置にノード O を配置する
4 \end{tikzpicture}
```



もし (0,0) の位置と 0.base の位置が一致するようにノードを配置したい場合は、次のように `anchor` オプションキーでアンカーを指定します。

指定した位置とノードのベースが一致するように配置する

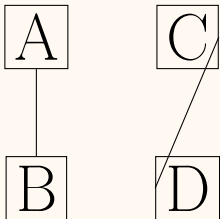
```
1 \begin{tikzpicture}
2   \draw (0,0) circle[radius=2pt];% 原点に目印
3   \node[draw, anchor=base] (0) at (0,0) {\Huge 0};% (0,0) の位置にノード 0 を配置する
4 \end{tikzpicture}
```



なお、ノードを線で結ぶと線の端点はノードのアンカーに接続されますが、このときはアンカーを省略しても `center` とは見なされず、TikZ が適切なアンカーを探し出してくれます。もちろん明示的に指定することもできます。

ノードを線で結ぶ

```
1 \begin{tikzpicture}
2   \node[draw] (A) at (0,2) {\Huge A};\node[draw] (C) at (2,2) {\Huge C};
3   \node[draw] (B) at (0,0) {\Huge B};\node[draw] (D) at (2,0) {\Huge D};
4   \draw (A) -- (B);% アンカーを指定しない
5   \draw (C.east) -- (D.west);% アンカーを指定する
6 \end{tikzpicture}
```



左の例ではノード間に線分を引くときにアンカーを指定していません。にもかかわらず2つのノードの `center` 間ではなく、上のノードの `south` と下のノードの `north` の間で線分が描画されています。右の例ではノード間に線分を引くときにアンカーを指定しています。指定したとおりに上のノードの `east` と下のノードの `west` の間で線分が描画されています。

5.7. オプション指定によるラベル配置

ラベルもノードの一種です。ノードを配置し、そこにテキストを表示するという考え方がラベル配置の基本になります。ところで、あるノードの隣にラベルを配置するという場合は、オプションを使って簡単にラベルを配置することもできます。

label オプションキーを `label=角度:テキスト` の形式で記述すると、テキストと表示位置の両方を同時に指定できます。label オプションキーで配置したラベルも1つのノードであることに注意してください。

基本

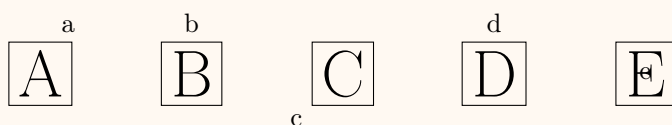
```
1 \begin{tikzpicture}
2   \node[draw] at (0,0) [label=right:B]{A};% A の右に B を配置
3 \end{tikzpicture}
```



この例ではラベル A の右にラベル B を label オプションキーを使って配置しています。角度には数値のほか、above や below left などの特別な角度および north などのアンカーを指定できます。

ラベル位置の指定

```
1 \begin{tikzpicture}
2   \node[draw] at (0,0) [label=70.5:a]{\Huge A};% A の 70.5 度方向に a を配置
3   \node[draw] at (2,0) [label=above:b]{\Huge B};% B の上に b を配置
4   \node[draw] at (4,0) [label=below left:c]{\Huge C};% C の左下に c を配置
5   \node[draw] at (6,0) [label=north:d]{\Huge D};% D の上に d を配置
6   \node[draw] at (8,0) [label=center:e]{\Huge E};% E の中央に e を配置
7 \end{tikzpicture}
```



上の例の4つめのノードの出力結果から分かるように、アンカー north とはラベル D のノードの north のことであることが分かります。また5つめのノードのように、アンカー center を指定することで、ノードの中央にラベルを配置することもできます。角度を省略すると above が指定されたと解釈されます。

label オプションキーに指定するテキストには注意が必要です。カッコ、コンマ、セミコロンなどの特殊な文字をテキストに含める場合は、引数全体を波カッコで囲む必要があります。

特殊な文字を含むテキスト

```
1 \begin{tikzpicture}
2   \node[draw] at (0,0) [label={above:(1,3)}]{A};% カッコ、コンマ、セミコロンを含む
3 \end{tikzpicture}
```



テキストにオプションを指定することもできます。

テキストのオプション

```

1 \begin{tikzpicture}
2   \node[draw] at (0,0) [label={[red, draw, circle] above:{\Huge a}}]{A};
3 \end{tikzpicture}

```



この例では [red, draw, circle] を指定することで、円形の輪郭を持つ赤いラベルを配置しています。

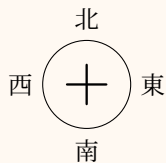
label オプションキーを複数指定することで、ラベルを複数配置することができます。

ラベルの複数配置

```

1 \begin{tikzpicture}
2   \node[draw, circle] at (0,0)
3     [label=north: 北, label=west: 西, label=south: 南, label=east: 東]
4     {\Huge $+}$; % ラベル東西南北を配置
5 \end{tikzpicture}

```



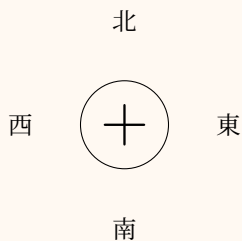
label distance オプションキーを指定することでノードとラベルの距離を指定することができます。

ラベルへの距離を指定

```

1 \begin{tikzpicture}[label distance=5mm]
2   \node[draw, circle] at (0,0)
3     [label=north: 北, label=west: 西, label=south: 南, label=east: 東]
4     {\Huge $+}$; % ラベル東西南北を距離を取って配置
5 \end{tikzpicture}

```



上の例では中央のノードから 5mm の距離を取って各ラベルを配置しています。tikzpicture 環境のオプションと

して `label distance` オプションキーを記述しているので、全ノードに対して `label distance=5mm` が指定されたことになります。

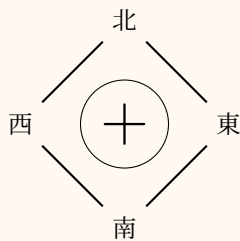
`label` オプションキーで配置したラベルもやはりノードの一つですので、ノード名をつけることができます。`name` オプションキーを使うことで命名できます。

ノードに命名

```

1 \begin{tikzpicture}[label distance=5mm]
2   \node[draw, circle] at (0,0)
3     [label={[name=kita] north: 北},
4      label={[name=nishi] west: 西},
5      label={[name=minami] south: 南},
6      label={[name=higashi] east: 東}]
7     {\Huge $+$}; % ラベルのノードに命名
8
9   \draw[thick] (kita) -- (nishi) -- (minami) -- (higashi) -- (kita);
10 \end{tikzpicture}

```



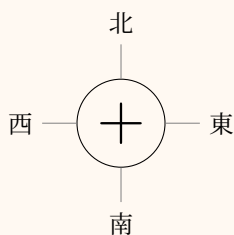
ノードとラベルの間を線で結ぶときは `label` オプションキーの代わりに `pin` オプションキーを指定することで、簡単に実現できます。

ラベルへ線を引く

```

1 \begin{tikzpicture}[label distance=5mm]
2   \node[draw, circle] at (0,0)
3     [pin=north: 北, pin=west: 西, pin=south: 南, pin=east: 東]
4     {\Huge $+$}; % ラベルへ線を引く
5 \end{tikzpicture}

```




`quotes` ライブラリを使えばラベルの配置がもっと簡潔にできます。事前に `quotes` ライブラリを読みこむ必要があります。

ラベルを簡潔に配置

```

1  \usetikzlibrary{quotes}
2  \begin{tikzpicture}
3      \node["A" below, draw] at (0,0) {B};% Bの下に Aを表示
4      \node["C" {below, red}, draw] at (2,0) {D};% Dの下に Cを表示
5  \end{tikzpicture}

```



`label` オプションキーを記述する代わりに、"ラベル" オプション の形式で記述します。ラベルについてのオプションが複数ある場合はそれらを波カッコで囲みます。

5.8. ラベルの複数行表示

ラベルを複数行にわたって表示するには、`node` のオプションとして `align` オプションキーを指定します。`align` オプションキーにはオプション値 `left` (左寄せ)、`center` (中央寄せ) または `right` (右寄せ) を指定します。

基本

```

1  \begin{tikzpicture}
2      \node[draw, align=left] (A)
3      {このテキストは\\任意の位置で改行し\\左寄せで表示しています。};% 左寄せ
4      \node[draw, align=center, anchor=south west] (B) at (A.south east)
5      {このテキストは\\任意の位置で改行し\\中央寄せで表示しています。};% 中央寄せ
6      \node[draw, align=right, anchor=south west] (C) at (B.south east)
7      {このテキストは\\任意の位置で改行し\\右寄せで表示しています。};% 右寄せ
8  \end{tikzpicture}

```

このテキストは 任意の位置で改行し 左寄せで表示しています。	このテキストは 任意の位置で改行し 中央寄せで表示しています。	このテキストは 任意の位置で改行し 右寄せで表示しています。
--------------------------------------	---------------------------------------	--------------------------------------

改行は \LaTeX での文法と同様 `\\` で行います。`align` オプションキーを指定しないと改行がされないことに注意しましょう。

一方、テキストの幅を指定することによって自動改行させることもできます。

テキストの幅を指定

```

1 \begin{tikzpicture}
2   \node[draw, text width=4cm]
3     {このテキストは指定した最大幅で折り返し表示しています。};
4 \end{tikzpicture}

```

このテキストは指定した最大幅で折り返し表示しています。

テキストの幅を `text width` オプションキーで指定します。このときは `align` オプションキーは必須ではなく、省略した場合のデフォルトは `left` になります。

英文を自動改行するときには、例えばハイフネーションをどうするかという問題があります。例えば左寄せの場合、`align` オプションキーへのオプション値として `left` または `flush left` を指定します。

改行の仕方の指定

```

1 \begin{tikzpicture}
2   \node[draw, text width=3cm, align=left] (A)
3     {(align=left)\Alice waited a little, half expecting to see it again, but it did not
4       appear,};% left
5   \node[draw, text width=3cm, align=flush left, anchor=north west] at (A.north east)
6     {(align=flush left)\Alice waited a little, half expecting to see it again, but it did not
7       appear,};% flush left
8 \end{tikzpicture}

```

(align=left)	(align=flush left)
Alice waited a little, half expecting to see it again, but it did not appear,	Alice waited a little, half expecting to see it again, but it did not appear,

上の例^{*1}の左が `left`、右が `flush left` で位置揃えしたものです。

`left` はハイフネーションによる単語分割をしてでも右端のバランスをとろうとします。`flush left` は単語分割をしません。

`right`、`flush right` や `center`、`flush center` も同様です。

`justify` は左右両端で位置揃えします。

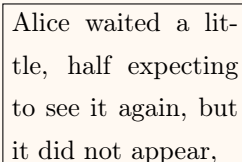
^{*1} Lewis Carroll, ALICE'S ADVENTURES IN WONDERLAND

左右両端で位置揃え

```

1 \begin{tikzpicture}
2   \node[draw, text width=3cm, align=justify]
3   {Alice waited a little, half expecting to see it again, but it did not appear,};% 左右両端で位置揃え
4 \end{tikzpicture}

```



以下に `align` オプションキーのオプション値をまとめます。

オプション値	説明
<code>left</code>	左寄せ、ハイフネーションによる単語分割あり
<code>flush left</code>	左寄せ、ハイフネーションによる単語分割なし
<code>center</code>	中央寄せ、ハイフネーションによる単語分割あり
<code>flush center</code>	中央寄せ、ハイフネーションによる単語分割なし
<code>right</code>	右寄せ、ハイフネーションによる単語分割あり
<code>flush right</code>	右寄せ、ハイフネーションによる単語分割なし
<code>justify</code>	左右両端で位置揃え

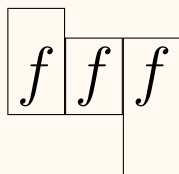
テキストの高さを指定するには `text height` オプションキー、`text depth` オプションキーを使います。

テキスト高さの指定

```

1 \begin{tikzpicture}
2   \node[draw] (A) {\Huge $f$};% 真ん中のラベル
3   \node[draw, anchor=base east, text height=1cm] at (A.base west) {\Huge $f$};% 左
4   \node[draw, anchor=base west, text depth=1cm] at (A.base east) {\Huge $f$};% 右
5 \end{tikzpicture}

```



上の例の真ん中が高さを指定しなかった場合、左が `text height` オプションキー、右が `text depth` オプションキーによる指定の結果です。

5.9. 線上に白抜きのラベルを配置する

ここでは線の上にラベルを配置し、同時にその周りを白抜きにするテクニックについて解説します。

例えば寸法を表示するときに、2点間を直線で結び、その途中に数値を表示したい場合があります。このとき線を途中で切る必要はありません。単にノードに少し大きさを持たせ、背景色を指定すれば良いだけです。

寸法を表示

```
1 \begin{tikzpicture}
2   \coordinate (A) at (0,0);\draw[fill] (A) circle[radius=1pt];
3   \coordinate (B) at (2,0);\draw[fill] (B) circle[radius=1pt];
4   \draw (A) ++(0,-0.1) -- ++(0,-0.1) coordinate (A1) -- ++(0,-0.1);
5   \draw (B) ++(0,-0.1) -- ++(0,-0.1) coordinate (B1) -- ++(0,-0.1);
6
7   \draw[<->] (A1) -- node[inner sep=2mm, fill=white] {2cm} (B1);
8 \end{tikzpicture}
```

第 6 章 平行移動、回転、拡大縮小編

この章では、図形全体あるいは図形の一部を、平行移動、回転または拡大縮小する方法を解説します。

6.1. 平行移動

水平方向の平行移動は `xshift` オプションキー、鉛直方向の平行移動は `yshift` オプションキーを指定します。両方指定する場合は `shift` オプションキーを指定します。

基本

```

1 \begin{tikzpicture}[every node/.style={draw, inner sep=5pt}]
2   \node (0,0) {指定なし};
3   \node[xshift=3cm] (0,0) {水平移動};% 右に 3cm 移動
4   \node[yshift=1cm] (0,0) {鉛直移動};% 上に 1cm 移動
5   \node[shift={(3cm,1cm)}] (0,0) {水平・鉛直移動};% 右に 3cm、上に 1cm 移動
6 \end{tikzpicture}

```

鉛直移動

水平・鉛直移動

指定なし

水平移動

例を見ると分かるように、いったん図を描画して最後に全体を平行移動するという挙動になっています。例えば「水平移動」ラベルを見ると、座標 (0,0) の位置にラベルをいったん配置し、そのあと水平右向きに 3cm 移動しています。結果的に (3cm, 0) の位置にラベルが配置されたことになります。

`xshift` オプションキーと `yshift` オプションキーではそれぞれの移動量を指定します。`shift` オプションキーでは `shift={(水平方向の移動量, 鉛直方向の移動量)}` の形式で指定します。

6.2. 回転

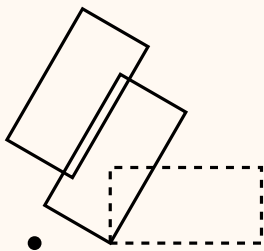
原点を中心とする回転は `rotate` オプションキー、任意の点を中心とする回転は `rotate around` オプションキーを指定します。キー

基本

```

1 \begin{tikzpicture}[very thick]
2   \draw[fill](0,0) circle[radius=2pt];% 原点
3
4   \draw[dashed] (1,0) rectangle (3,1);% 回転前
5   \draw[rotate=60] (1,0) rectangle (3,1);% 原点を中心に 30 度回転
6   \draw[rotate around={60:(1,0)}] (1,0) rectangle (3,1);% (1,0) を中心に 30 度回転
7 \end{tikzpicture}

```



`rotate` オプションキーでは回転角を指定します。`rotate around` オプションキーでは `rotate around={角度:(回転の中心点の座標)}` の形式で指定します。

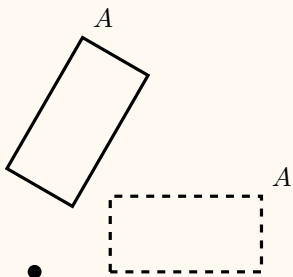
ノードに対して回転を行うと、ノードの位置は回転に追従しますが、ノード自身は傾いたりはしません。

ノードへの回転

```

1 \begin{tikzpicture}[very thick]
2   \draw[fill](0,0) circle[radius=2pt];% 原点
3
4   \draw[dashed] (1,0) rectangle (3,1) node[above right] {$A$};
5   \draw[rotate=60] (1,0) rectangle (3,1) node[above right] {$A$};
6 \end{tikzpicture}

```



6.3. 拡大縮小

原点を中心とする拡大縮小は `scale` オプションキー、任意の点を中心とする拡大縮小は `scale around` オプションキーを指定します。

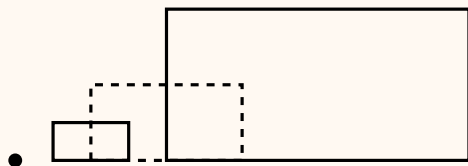
`scale` オプションキーでは倍率を指定します。

基本

```

1 \begin{tikzpicture}[very thick]
2   \draw[fill](0,0) circle[radius=2pt];% 原点
3   \draw[dashed] (1,0) rectangle (3,1);% 拡大縮小前
4   \draw[scale=2] (1,0) rectangle (3,1);% 原点を中心として2倍に拡大
5   \draw[scale=0.5] (1,0) rectangle (3,1);% 原点を中心として半分に縮小
6 \end{tikzpicture}

```



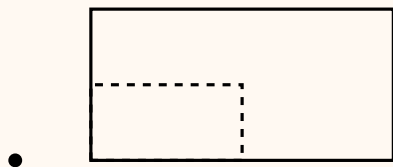
`scale around` オプションキーでは `scale around={倍率:(拡大縮小の中心点の座標)}` の形式で指定します。

中心を指定した拡大縮小

```

1 \begin{tikzpicture}[very thick]
2   \draw[fill](0,0) circle[radius=2pt];% 原点
3   \draw[dashed] (1,0) rectangle (3,1);% 拡大縮小前
4   \draw[scale around={2:(1,0)}] (1,0) rectangle (3,1);% (1,0)を中心として2倍に拡大
5 \end{tikzpicture}

```



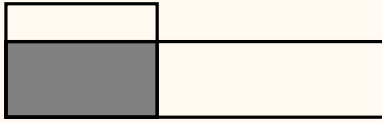
水平方向のみおよび鉛直方向のみの拡大縮小は、それぞれ `xscale` オプションキーおよび `yscale` オプションキーを指定します。

水平方向のみおよび鉛直方向のみの拡大縮小

```

1 \begin{tikzpicture}[very thick]
2   \draw[fill=gray] (0,0) rectangle (2,1);% 拡大縮小前
3   \draw[xscale=2.5] (0,0) rectangle (2,1);% 原点を中心として水平方向に拡大
4   \draw[yscale=1.5] (0,0) rectangle (2,1);% 原点を中心として鉛直方向に拡大
5 \end{tikzpicture}

```



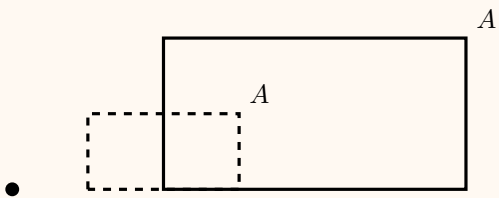
ノードに対して拡大縮小を行うと、ノードの位置は拡大縮小に追従しますが、ノード自身は拡大縮小しません。

ノードへの拡大縮小

```

1 \begin{tikzpicture}[very thick]
2   \draw[fill](0,0) circle[radius=2pt];% 原点
3   \draw[dashed] (1,0) rectangle (3,1) node[above right] {$A$};
4   \draw[scale=2] (1,0) rectangle (3,1) node[above right] {$A$};
5 \end{tikzpicture}

```



通常 `scale` オプションキーは `tikzpicture` 環境のオプションとして指定する人が多いようです。その場合描画した線分などのすべてが拡大縮小の対象となります。もちろんそのときもノード自身の大きさは対象外です。

第 7 章 作図風コマンド編

TikZ では 2 つの曲線の交点を求めたり、任意の直線へ垂線を引くなどの機能がライブラリとして用意されています。またそれらのうちの一部の機能はベクトルの概念を使って作図することになります。この章では、ベクトル演算を援用するなどの、幾何学的な考え方で作図を行う方法を解説します。

7.1. 鉛直線と水平線の交点の座標

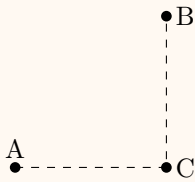
与えられた 2 点のノード名を A、B としたとき、A からの水平線と、B からの鉛直線の交点は $(A|B)$ で、A からの鉛直線と、B からの水平線の交点は $(A|-B)$ で取得できます。取得結果はいずれも座標であり、コマンド内の座標指定に使用することができます。

基本

```

1 \begin{tikzpicture}
2   \coordinate (A) at (0,0);
3   \coordinate (B) at (2,2);
4   \fill (A) circle[radius=2pt] node[above] {A};
5   \fill (B) circle[radius=2pt] node[right] {B};
6
7   \fill (A -| B) circle[radius=2pt] node[right] {C};% 交点
8   \draw[dashed] (A) -- (A -| B) -- (B);% 交点を經由する直線
9 \end{tikzpicture}

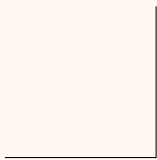
```



上の例ではノード名 A、B を使用して (A|-B) のように記述していますが、ノード名の代わりに座標を直接指定することもできます。

基本

```
1 \begin{tikzpicture}
2   \draw (0,0) -- ({(0,0)} -| {(2,2)}) -- (2,2); % 直接座標を指定
3 \end{tikzpicture}
```



7.2. 任意のパスの交点

`intersections` ライブラリを使うことで、任意の曲線の交点を求めることができます。

交点を扱うには次の手順を踏むことになります。

1. 2つのパスを定義し、それらのパスに命名する。
2. 2つのパスの交点を求め、その交点にノード名をつける。

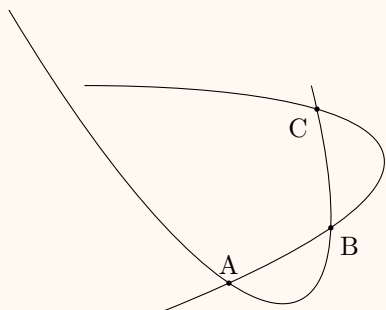
以下に例を示します。

基本

```
1 \usetikzlibrary{intersections}
2 \begin{tikzpicture}
3   \draw[name path=c1] (-2,0) .. controls (3, 2) and (1, 3) .. (-3,3);
4   \draw[name path=c2] (-4,4) .. controls (-1,-1) and (1, -1) .. (0,3);
5   \fill[name intersections={of= c1 and c2, by={A, B, C}}] % 交点を求め命名する
6   (A) circle[radius=1pt] node[above] {A}
7   (B) circle[radius=1pt] node[below right] {B}
8   (C) circle[radius=1pt] node[below left] {C};
```



```
9 \end{tikzpicture}
```



3 行目と 4 行目では 3 次ベジエ曲線を描画しています。それと同時に `name path` オプションキーでパスに名前をつけています。5 行目では `name intersections` オプションキーで 2 つのパスの交点を求め、それらの交点に対しノード名をつけています。例では引き続き各交点に黒点を描画しています。`name intersections` オプションキーのオプション値は次のように記述します。`of= パス 1 and パス 2` の形式で 2 つのパスを指定します。`by={交点 1 名, 交点 2 名, ...}` の形式で各交点のノード名を定義します。ノード名の定義 `by` を省略すると、TikZ が自動的に `intersection-1`, `intersection-2`, ... というノード名をつけます。

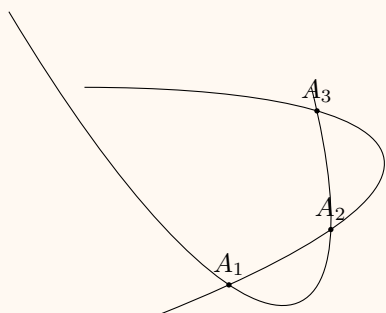
`by` オプションキーで指定できるのはノード名のみであり、それぞれのノード名をどの交点につけるかまでは指定することはできません。

交点の数があらかじめ分かっている場合は `by` オプションキーでノード名を定義できますが、分からない場合は TikZ に命名を任せるしかありません。その際、交点の数を知りたいこともあるでしょう。`total=マクロ` を記述すると指定したマクロに交点の数が格納されます。ここでマクロとは `\` で始まる変数名のことと思って差し支えありません。あとは `\foreach` コマンドを併用して、ループを回すことになるでしょう。

交点への自動命名では `intersection-n` の形のノード名となりますが、`name=接頭辞` を指定すると `接頭辞-n` の形のノード名にすることができます。

交点数が未知の場合

```
1 \usetikzlibrary{intersections}
2 \begin{tikzpicture}
3   \draw[name path=c1] (-2,0) .. controls (3, 2) and (1, 3) .. (-3,3);
4   \draw[name path=c2] (-4,4) .. controls (-1,-1) and (1, -1) .. (0,3);
5
6   % 交点を求め自動命名する
7   \fill[name intersections={of= c1 and c2, name=is, total=\total}]
8   \foreach \n in {1,...,\total}{(is-\n) circle[radius=1pt] node[above] {$A_{\n}$}};
9 \end{tikzpicture}
```



この例ではマクロ `\total` に交点数、今の場合 3、が格納されています。また交点のノード名は `is-1`、`is-2`、`is-3` となります。

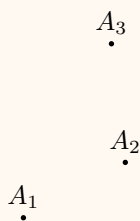
2 つの曲線は描画したくないが、交点だけは求めたいという場合は、`\draw` コマンドの代わりに `\path` コマンドを使います。

曲線を描画しない場合

```

1  \usetikzlibrary{intersections}
2  \begin{tikzpicture}
3      % 曲線を描画しない
4      \path[name path=c1] (-2,0) .. controls (3, 2) and (1, 3) .. (-3,3);
5      \path[name path=c2] (-4,4) .. controls (-1,-1) and (1, -1) .. (0,3);
6
7      \fill[name intersections={of= c1 and c2, name=is, total=\total}]
8      \foreach \n in {1,...,\total}{(is-\n) circle[radius=1pt] node[above] {$A_{\n}$}};
9  \end{tikzpicture}

```



7.3. 位置ベクトルの和とスカラー倍を使った座標指定

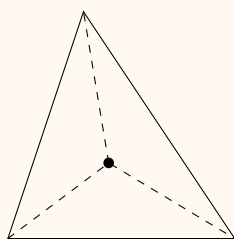
数学では点の座標を指定する代わりに原点からの位置ベクトルを指定することがあります。そして2つのベクトルの和、差やスカラー倍から得られる位置ベクトルを使って、新しい座標を得ることができます。calc ライブラリを使えば、この考え方を使った座標指定を行うことができます。

基本

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3    \draw (0,0) coordinate (A) -- (3,0) coordinate (B) -- (1,3) coordinate (C) -- cycle;
4
5    % 重心
6    \fill ($1/3*(A) + 1/3*(B) + 1/3*(C)$) circle[radius=2pt] coordinate (D);
7    \draw[dashed] (A) -- (D) -- (B) (D) -- (C);
8  \end{tikzpicture}

```



$(\$...\$)$ の中に計算式を記述すると、結果として座標が得られます。計算式の中にノードを指定すると、原点 (0,0) からの位置ベクトルとみなされます。

上の例では三角形の重心に黒点を描画しています。重心は $(\vec{OA} + \vec{OB} + \vec{OC})/3$ で求められますので、それに対応して $(\$1/3*(A) + 1/3*(B) + 1/3*(C)$)$ で重心の座標を表現しています。

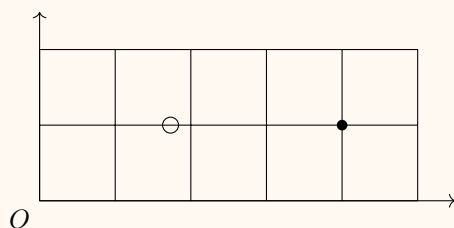
計算式の中にノード名を記述するときは (A) のように (...) で囲むことに注意してください。ノード名の代わりに $(\$2*(2,0) - (0,1)$)$ のように座標を直接指定することもできます。 $(\$2*\cos(30)*(1,0) + 2*\sin(30)*(0,1)$)$ のように関数を使用することもできます。

様々な指定方法

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3    \draw (0,0) grid (5,2);
4    \draw[<->] (0,2.5) -- (0,0) node[below left] {$0$} -- (5.5,0);
5
6    \fill ($2*(2,1) - (0,1)$) circle[radius=2pt]; % 右の黒点
7    \draw ($2*\cos(30)*(1,0) + 2*\sin(30)*(0,1)$) circle[radius=3pt]; % 左の白点
8  \end{tikzpicture}

```



なお、和、差とスカラー倍を併用するときは、先にスカラー倍、その次に和、差を記述することに注意してください。 $(\frac{1}{3}*(A) + \frac{1}{3}*(B) + \frac{1}{3}*(C))$ は正しい文ですが、 $(\frac{1}{3}*((A) + (B) + (C)))$ はエラーになります。数学で言うところの分配則は認識してくれません。

7.4. 内分点・外分点

与えられた 2 点 A と B の内分点、外分点を求め、その座標を取得することができます。このとき `calc` ライブラリを使います。

AB 間を $t:1-t$ で内分・外分する場合は、 $(\$ (A \text{ の座標})!t!(B \text{ の座標})\$)$ の形式で記述します。 $(\$...\$)$ で囲むことに注意しましょう。 $0 < t < 1$ の場合は内分、 $t < 0$ または $1 < t$ の場合は外分になります。

ところで、`xcolor` での色の比率の指定の仕方とは次の意味で違いがあることに注意しましょう。内分・外分の比率の単位はパーセントではありません。内分の場合 t が大きければ大きいほど A から遠ざかります。 t は負数や 1 より大きい数を取ることもできます。

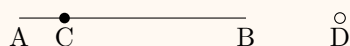
計算式の中にノード名を記述するときは (A) のように (...) で囲むことに注意してください。座標を直接指定することもできます。 $(\$ (0,0)!sqrt(2)!(3,0)\$)$ のように関数を指定することもできます。

基本

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3    \coordinate[label=below:A] (A) at (0,0);
4    \coordinate[label=below:B] (B) at (3,0);
5    \draw (A) -- (B);
6
7    % 0.2:0.8 の内分点
8    \fill ($ (A)!0.2!(B)$) circle[radius=2pt] node[below] {C};
9    % sqrt(2):1-sqrt(2) の外分点
10   \draw ($ (0,0)!sqrt(2)!(3,0)$) circle[radius=2pt] node[below] {D};
11 \end{tikzpicture}

```



$(\$ (A \text{ の座標})!t!\text{角度}:(B \text{ の座標})\$)$ の形式で記述すると、先に指定した角度で回転してから内分・外分を行います。

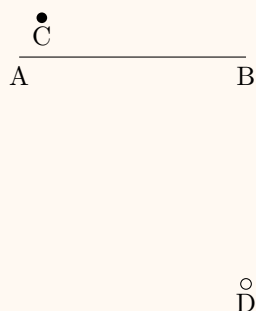
より正確には、A を中心として B を指定した角度だけ仮想的に回転します。角度は反時計回りを正とします。そのうえで指定した比率に従って内分・外分を行います。

回転してから内分・外分

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \coordinate[label=below:A] (A) at (0,0);
4      \coordinate[label=below:B] (B) at (3,0);
5      \draw (A) -- (B);
6
7      % 60 度回転して内分
8      \fill ($(A)!.2!60:(B)$) circle[radius=2pt] node[below] {C};
9      % atan(-1) (= -45)度回転して外分
10     \draw ($(0,0)!\sqrt{2}!atan(-1):(3,0)$) circle[radius=2pt] node[below] {D};
11 \end{tikzpicture}

```



上の例のラベル C のノードの場合は、まず A を中心として B を 60 度だけ仮想的に回転します。そして A と（仮想的な）B との間を 0.2 : 0.8 で内分し、そこに黒点を描画しています。

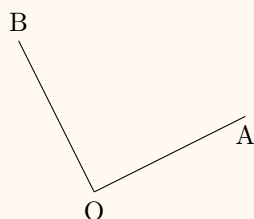
よく使う場面としては、例えば与えられた直線への垂直線などが挙げられるでしょう。

与えられた直線への垂直線

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \coordinate[label=below:O] (O) at (0,0);
4      \coordinate[label=below:A] (A) at (2,1);
5      \draw (O) -- (A);
6
7      \draw (O) -- ($(O)!\sqrt{2}!90:(A)$) node[above] {B}; % 垂直線
8  \end{tikzpicture}

```



7.5. 指定した 2 点間の指定した距離の座標

与えられた 2 点 A と B を通り、A から指定距離だけ離れた座標を取得することができます。このとき `calc` ライブラリを使います。

$(\$ (A \text{ の座標})! \text{距離}! (B \text{ の座標}) \$)$ の形式で記述します。すると A から B に向かう直線上の、A から指定した距離の点の座標を取得できます。 $(\$ \dots \$)$ で囲むことに注意しましょう。距離には `1cm` や `10pt` などのように必ず単位を付けます。単位をつけないと 7.4 節のように、比率を指定したと解釈されてしまいます。

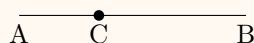
計算式の中にノード名を記述するときは (A) のように (...) で囲むことに注意してください。座標を直接指定することもできます。

基本

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3    \coordinate[label=below:A] (A) at (0,0);
4    \coordinate[label=below:B] (B) at (3,0);
5    \draw (A) -- (B);
6
7    \fill ($ (A)!30pt!(B) $) circle[radius=2pt] node[below] {C}; % A から 30pt の距離
8  \end{tikzpicture}

```



$(\$ (A \text{ の座標})! \text{距離}! \text{角度}! (B \text{ の座標}) \$)$ の形式で記述すると、先に指定した角度で回転してから、指定した距離の座標を取得します。より正確には、A を中心として B を指定した角度だけ仮想的に回転します。角度は反時計回りを正とします。そのうえで指定した距離だけ離れた点の座標を取得します。

回転してから指定距離離れた点を取得

```

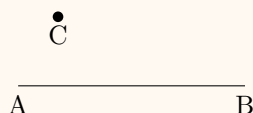
1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3    \coordinate[label=below:A] (A) at (0,0);

```

```

4 \coordinate[label=below:B] (B) at (3,0);
5 \draw (A) -- (B);
6
7 % 60 度回転してから A から 30pt の距離
8 \fill ($(A)!30pt!60:(B)$) circle[radius=2pt] node[below] {C};
9 \end{tikzpicture}

```



上の例のラベル C のノードの場合は、まず A を中心として B を 60 度だけ仮想的に回転します。そして A から (仮想的な) B に向かって 30pt 離れた点に黒点を描画しています。

よく使う場面としては、例えば与えられた直線への垂直線などがあるでしょう。

与えられた直線への垂直線

```

1 %\usetikzlibrary{calc}
2 \begin{tikzpicture}
3 \coordinate[label=below:O] (O) at (0,0);
4 \coordinate[label=below:A] (A) at (2,1);
5 \draw (O) -- (A);
6
7 % 垂直線
8 \draw (O) -- node[sloped, above] {1cm} ($(O)!1cm!90:(A)$) circle[radius=2pt, fill];
9 \end{tikzpicture}

```



7.6. 垂直線との交点

点 P と直線 AB が与えられたとき、P から AB への垂直線と AB との交点を取得することができます。このとき calc ライブラリを使います。

$(\$ (A \text{ の座標})! (P \text{ の座標})! (B \text{ の座標}) \$)$ の形式で記述します。すると A と B を通る直線と、P から AB への垂直線との交点の座標を取得できます。 $(\$ \dots \$)$ で囲むことに注意しましょう。

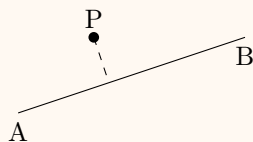
計算式の中にノード名を記述するときは (A) のように (...) で囲むことに注意してください。座標を直接指定することもできます。

基本

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \coordinate[label=below:A] (A) at (0,0);
4      \coordinate[label=below:B] (B) at (3,1);
5      \coordinate[label=above:P] (P) at (1,1);
6      \draw (A) -- (B);
7      \fill (P) circle[radius=2pt];
8
9      \draw[dashed] (P) -- ($ (A)!(P)!(B) $); % P から AB への垂線
10 \end{tikzpicture}

```



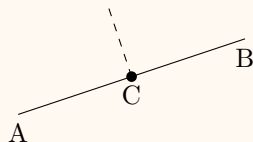
逆に交点が先に与えられていて、それを通る垂線を引きたい場合は 7.5 節の方法を使います。

与えられた点を通る垂線

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \coordinate[label=below:A] (A) at (0,0);
4      \coordinate[label=below:B] (B) at (3,1);
5      \coordinate[label=below:C] (C) at ($ (A)!0.5!(B) $);
6      \draw (A) -- (B);
7      \fill (C) circle[radius=2pt];
8
9      \draw[dashed] (C) -- ($ (C)!1cm!90:(B) $); % C からの垂線
10 \end{tikzpicture}

```



7.7. 平行線を引く

直線 AB が与えられたとき、それと平行な直線を引くことができます。これは 7.3 節の応用です。

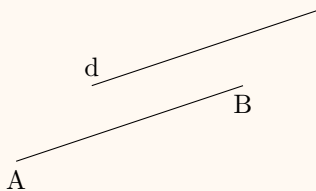
まず直線の平行移動量がベクトルとして与えられていれば、次のように作図できます。

平行移動量がベクトルとして与えられている場合

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \coordinate[label=below:A] (A) at (0,0);
4      \coordinate[label=below:B] (B) at (3,1);
5      \coordinate[label=above:d] (d) at (1,1);
6      \draw (A) -- (B);
7
8      \draw ($(A)+(d)$) -- ($(B)+(d)$);
9  \end{tikzpicture}

```



上の例では、ベクトル d を平行移動量として、ベクトル和を使って描画しています。

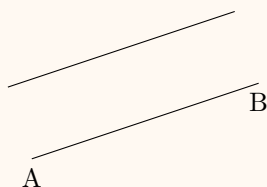
平行移動量がベクトルとして与えられていない場合は、事前にベクトルを求めることになります。例えば線分 AB と垂直方向に、指定した距離だけ移動させるような、平行移動のベクトルを算出するなどがあるでしょう。

法線方向に移動した線分

```

1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \coordinate[label=below:A] (A) at (0,0);
4      \coordinate[label=below:B] (B) at (3,1);
5      \draw (A) -- (B);
6
7      % 平行移動量を算出
8      \coordinate (d) at ($(A)!1cm!90:(B) - (A)$); % AB から 1cm 隔てる
9
10     \draw ($(A)+(d)$) -- ($(B)+(d)$);
11 \end{tikzpicture}

```



上の例の平行移動量を算出している部分に注目しましょう。まず A を中心として B を 90 度だけ仮想的に回転させ、 A から仮想的な B に向かって 1cm だけ移動します。その点から A を引くと、求める平行移動のベクトルとなります。

7.8. 角度記号を描く

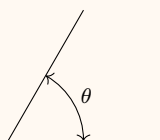
`angles` ライブラリを使うと角度記号を簡単に描画できます。使い方をまず見てみましょう。

基本

```

1 \usetikzlibrary{angles, quotes}
2 \begin{tikzpicture}
3   \draw (2,0) coordinate (A) -- (0,0) coordinate (B) -- (60:2) coordinate (C);
4
5   \draw pic["$\theta$", draw, <->, font=\footnotesize, angle eccentricity=1.2, angle
      radius=1cm] {angle=A--B--C};% 角
      度
6 \end{tikzpicture}

```



`\draw pic {angle=A--B--C}` の書式が基本です。このとき B の角度を表す記号が描画されます。A、B および C はノード名である必要があり、座標の直接指定はできません。

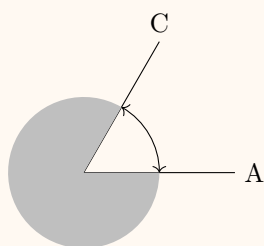
3 点を A--B--C の順序で記述した場合、直線 BA 上の点を始点とし、反時計周りで回り、直線 BC 上の点を終点とするように、角度記号の円弧が描画されます。

円弧の描画のしかた

```

1 \usetikzlibrary{angles}
2 \begin{tikzpicture}
3   \draw (2,0) coordinate (A) node[right] {A} -- (0,0) coordinate (B) -- (60:2) coordinate (C)
      node[above] {C};
4
5   \draw pic[draw, <->, angle radius=1cm] {angle=A--B--C};% 角度
6   \draw pic[fill=lightgray, angle radius=1cm] {angle=C--B--A};% 角度
7 \end{tikzpicture}

```



上の例では `angle=A--B--C` のほうが内角、`angle=C--B--A` のほうが外角を描画しています。

オプションには次のものがあります。`draw` オプションキーを指定すると角度を表す円弧が描画されます。逆に言え

ば `draw` オプションキーを指定しないと円弧が描画されません。塗りつぶしの `fill` オプションキーや、矢印の `<->` も使えます。`angle radius` オプションキーで円弧の半径を指定できます。

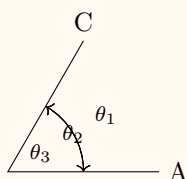
角度を表すラベルを描画する場合は `quotes` ライブラリを読み込む必要があります。`pic["ラベル"]` の形式でラベルを指定します。ラベルの大きさは `font` オプションキーで指定します。ラベルの配置位置は `angle eccentricity` オプションキーで指定します。以下に `angle eccentricity` オプションキーの例を示します。

ラベルの配置位置

```

1  \usetikzlibrary{angles, quotes}
2  \begin{tikzpicture}
3      \draw (2,0) coordinate (A) node[right] {A} -- (0,0) coordinate (B) -- (60:2) coordinate(C)
        node[above] {C};
4
5      \draw pic["$\\theta_1$", draw, <->, font=\footnotesize, angle eccentricity=1.5, angle
        radius=1cm] {angle=A--B--C};
6      \draw pic["$\\theta_2$", draw, <->, font=\footnotesize, angle eccentricity=1, angle
        radius=1cm] {angle=A--B--C};
7      \draw pic["$\\theta_3$", draw, <->, font=\footnotesize, angle eccentricity=0.5, angle
        radius=1cm] {angle=A--B--C};
8  \end{tikzpicture}

```



`angle eccentricity` オプションキーのオプション値が大きいと円弧の外側、小さいと内側に配置されることが分かります。1 で円弧と重なります。

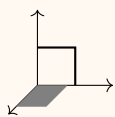
直角記号もあります。`angle` の代わりに `right angle` を使います。

直角記号

```

1  \usetikzlibrary{angles}
2  \begin{tikzpicture}
3      \coordinate (O) at (0,0,0);
4      \draw[<-] (1,0,0) coordinate (A) -- (O);
5      \draw[<-] (0,0,1) coordinate (B) -- (O);
6      \draw[<-] (0,1,0) coordinate (C) -- (O);
7
8      \draw pic [fill=gray,angle radius=4mm] {right angle = A--O--B};% AOB 間の直角
9      \draw pic [draw,thick] {right angle = A--O--C};% AOC 間の直角
10 \end{tikzpicture}

```



7.9. 与えられた中心を持ち、与えられた点を通る円

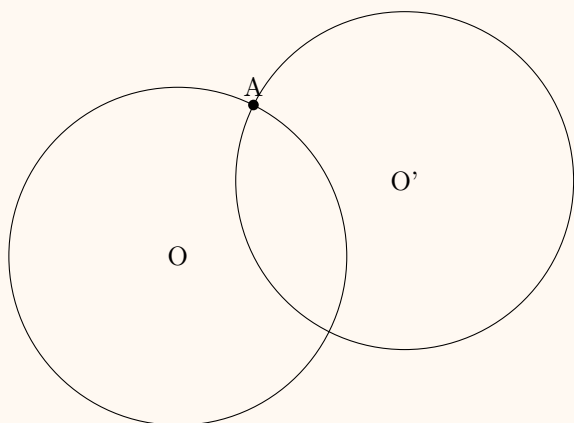
`through` ライブラリを使うと、与えられた中心を持ち、与えられた点を通る円を描くことができます。使い方をまず見てみましょう。次の例では与えられた点 $A(1,2)$ を通り、原点を中心とする円、および A を通り、 $(3,1)$ を中心とする円を作図しています。

基本

```

1  \usetikzlibrary{through}
2  \begin{tikzpicture}
3    \fill (1,2) circle[radius=2pt] coordinate[label=above:A] (A);
4
5    % (0,0) を中心とする A を通る円
6    \node[draw, circle through=(A)] at (0,0) {O};
7    % (3,1) を中心とする (1,2) を通る円
8    \node[draw, circle through={(1,2)}] at (3,1) {O'};
9  \end{tikzpicture}

```



例を見ると分かるように、円と言っても実はノードの輪郭としての円であることが分かります。

`node` の `circle through` オプションキーに経由したい座標を指定します。上の例のようにノード名による指定でも、座標の直接指定でも構いません。

7.10. 座標計算を連続して記述する

calc ライブラリを使った座標計算を連続して記述することができます。

例えば与えられた線分 AB の垂直二等分線を引くにはどうしたら良いでしょうか？それには AB の中点 (0.5 : 0.5 の内分点) C を求め、C を中心として 90 度回転させたある長さの線分を引けば良いですね。そこで内分点を求める座標計算と、90 度回転させてある距離だけ伸ばした点の座標計算をするのですが、それを一文で書いてみます。

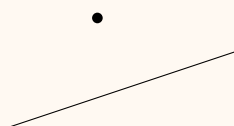
答は $(\$ (A)!.5!(B)!1cm!90:(B)\$)$ となります。評価は前から行われます。すなわち、最初に $(\$ (A)!.5!(B)\$)$ が評価されます。この座標を説明の便宜上 C としましょう。次に $(\$ (C)!1cm!90:(B)\$)$ が評価されます。結果として得られる座標は、線分 AB の中点から垂直方向に 1cm だけ離れた点になります。

2 つ連続した座標計算

```

1 \usetikzlibrary{calc}
2 \begin{tikzpicture}
3   \draw (0,0) coordinate (A) -- (3,1) coordinate (B);
4
5   \fill ($ (A)!.5!(B)!1cm!90:(B)\$) circle[radius=2pt]; % AB の中点から垂直方向に 1cm だけ離れた点
6 \end{tikzpicture}

```



3 つ以上連続した座標計算もちろんできます。

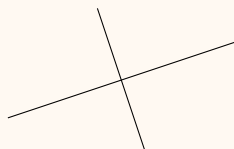
それでは垂直二等分線を引いてみましょう。

垂直二等分線 1

```

1 \usetikzlibrary{calc}
2 \begin{tikzpicture}
3   \draw (0,0) coordinate (A) -- (3,1) coordinate (B);
4
5   \draw ($ (A)!.5!(B)!1cm!90:(B)\$)
6         -- ($ (A)!.5!(B)!-1cm!90:(B)\$); % 垂直二等分線
7 \end{tikzpicture}

```



さて、これを見ると、座標計算の連続記述には限界があると思う方もいるかもしれません。一文で書けてすっきりしている反面、可読性には若干の難があります。またこの例のように、垂直二等分線の 2 つの端点を求めるのに、似たような座標記述を 2 つ書いています。つまり一文で簡潔には書けないような座標もあるということです（少なくとも

筆者にはもっと良い方法が思いつきませんでした)。

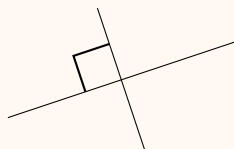
座標計算の連続記述と、適宜ノード名をつけながらの座標記述をうまく使い分けると、すっきりかつ可読性のある記述ができるようになるでしょう。

垂直二等分線 2

```

1  \usetikzlibrary{calc, angles}
2  \begin{tikzpicture}
3      \draw (0,0) coordinate (A) -- (3,1) coordinate (B);
4
5      \draw ($(A)!.5!(B)$) coordinate (C) % 中点
6          ($(C)!1cm!90:(B)$) coordinate (D) % 一方の端点
7          -- ($(D)!2!(C)$) % 垂直二等分線
8          pic [draw, thick] {right angle = A--C--D};
9  \end{tikzpicture}

```



7.11. レジスタ

calc ライブラリには描画を補助するためのレジスタがあります。レジスタとは値を一時的に格納する変数のようなものです (レジスタという仕組みは calc ライブラリに限らず \TeX 自体にあるものです)。レジスタはより複雑な計算を行う場合に役立ちます。

TikZ で使用するときの基本的な構文は

`\path let レジスタへの代入文, レジスタへの代入文, ... in オペレーション列`

となります。`\path` コマンド (`\draw` コマンドなども含む) のあとに `let` を使ってレジスタに値を格納します。続けて `in` のあとに描画のためのオペレーション列を記述しますが、そこにレジスタを含めることで、その値を使うことができます。なお、レジスタの有効範囲は一つのコマンド内に限ります。

レジスタの使用例

```

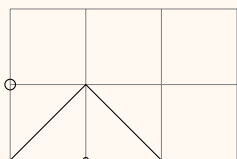
1  \usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \draw[help lines] (0,0) grid (3,2);
4
5      \draw let \p1 = (1,1), \p2 = (2,0), \n1=1, \n2={1pt} in

```

```

6      % レジスタの値を使って線分を引く
7      (0,0) -- (\p1) -- (\p2)
8      % (\p1 の x 成分, 0) を中心とする半径 \n1 の円を描く
9      (\x1,0) circle[radius=\n1 pt]
10     % (0, \p1 の y 成分) を中心とする半径(\n1 + 1pt) の円を描く
11     (0,\y1) circle[radius={\n1 + 1pt}];
12 \end{tikzpicture}

```



`\p` で始まるレジスタには座標が格納されます。上の例ではレジスタ `\p1` には (1,1) が、レジスタ `\p2` には (2,0) が格納されています。それらを使って7行目で線分を引いています。

`\n` で始まるレジスタには数値が格納されます。上の例ではレジスタ `\n1` には 1 が、レジスタ `\n2` には 1pt が格納されています。これらは9行目と11行目で円の半径として使われています。`\n1` のようにただの数値を格納することもできますし、`\n2` のように単位つきの数値を格納することもできます。

`\x` や `\y` で始まるレジスタは、`\p` で始まるレジスタに格納された座標の x 成分と y 成分を返します。正確には、レジスタ `\x1` にはレジスタ `\p1` の x 成分が、レジスタ `\y1` には `\p1` の y 成分が入っています。これらには値を任意に格納することはできません。上の例では9行目と11行目で円の中心の座標指定に使われています。

レジスタ名は `\p1`、`\n1` などのように、`\p`、`\n` の後に整数を付しますが、整数ではなく任意の名前をつけることもできます。その際は波カッコで囲み、`\p{point}`、`\p{A}` などと記述します。

7.12. 直線に接する円を描く

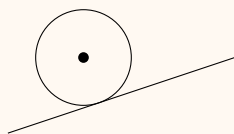
直線 AB と点 P が任意に与えられたときの、P を中心とし AB と接する円を作図してみましょう。これには7.11節で解説しているレジスタを使用します。

直線に接する円

```

1  %\usetikzlibrary{calc}
2  \begin{tikzpicture}
3      \draw (0,0) coordinate (A) -- (3,1) coordinate (B);
4      \fill (1,1) circle[radius=2pt] coordinate (P);
5
6      \draw let \p1 = ($(A)!(P)!(B) - (P)$), % 半径を表すベクトル
7              \n1 = {veclen(\x1,\y1)} % 半径
8              in (P) circle[radius=\n1]; % 求めた半径の円
9  \end{tikzpicture}

```



`vecLen(a, b)` は数学関数 (9.2 節参照) であり $\sqrt{a^2 + b^2}$ を返します。

`(A)!(P)!(B)` は P から AB に下ろした垂線と AB との交点の座標ですので、`(A)!(P)!(B) - (P)` はその交点と P との間を結ぶベクトルになります。これが求める円の半径を表すベクトルになります。それから半径を求め、レジスタ `\n1` に格納しています。

第 8 章 スタイル編

この章ではスタイルについて解説します。

スタイルとは特定のオプション指定を指し、オリジナルのスタイルを定義することができます。指定するオプションは複数でも構いません。一度スタイルを定義してしまえば、あとはそのスタイル名を個々の図要素のオプションとして記述するだけで、特定のオプション指定を行ったことと同等になります。

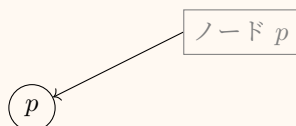
8.1. スタイル定義

スタイル定義は、**スタイル名/.style={オプション指定}** の形式で記述します。

基本

```

1 \begin{tikzpicture}[annot/.style={gray, draw}] %注釈のスタイル
2   \node[draw, circle] (p) at (0,0) {$p$};
3   \draw[<-] (p) -- (2,1) node[annot, right] {ノード $p$};
4 \end{tikzpicture}
```



この例では `gray`, `draw` の 2 つのオプション指定に `annot` という名前でスタイル定義しています。このようにスタイル定義に指定するオプションは複数でも構いません。

このスタイルを使用している箇所が `node[annot, right]` になります。グレーかつ枠線つきでラベルが描画されることが分かります。

`tikzpicture` 環境でスタイルを定義すると、有効範囲はその `tikzpicture` 環境内となります。文書全体で有効なスタイルを定義する場合は、プリアンブル部に次のように記述します。

グローバルなスタイル

```

1  % プリアンブル部
2  \tikzset{annot/.style={gray, draw}}

```

一方、一つの `tikzpicture` 環境内であっても、さらに有効範囲を限定したい場合は `scope` 環境内でスタイルを定義します。

ローカルなスタイル

```

1  \begin{scope}[annot/.style={gray, draw}]
2
3  \end{scope}

```

スタイルを使用するときに、同時に別のオプションを追加で指定することもできます。例えば `node[annot, right]` では、スタイル `annot` とオプション `right` の両方が適用されます。もし、その結果互いに矛盾したオプション指定になるならば、あとに記述した方が有効になります。

矛盾した指定

```

1  \begin{tikzpicture}[annot/.style={gray, draw}] % 注釈のスタイル
2      \node[annot, black] at (0,0) {$p$}; % black が後
3      \node[black, annot] at (1,0) {$q$}; % annot が後
4  \end{tikzpicture}

```



この例を見ると、`\node[annot, black]` では `black` をあとに記述しているので、黒が有効です。`\node[black, annot]` では `annot` をあとに記述しているので、グレーが有効です。一方、どちらでも `draw` は活きているのでいずれでも輪郭は描画されます。

8.2. 引数つきスタイル

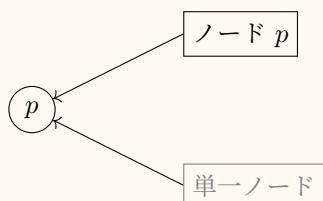
引数をとるスタイルを定義することができます。例を見てみましょう。

基本

```

1  \begin{tikzpicture}[annot/.style={color=#1, draw},
2                      annot/.default=gray] % 注釈のスタイル
3      \node[draw, circle] (p) at (0,0) {$p$};
4      \draw[<-] (p) -- (2,1) node[annot=black, right] {ノード $p$};
5      \draw[<-] (p) -- (2,-1) node[annot, right] {単一ノード};
6  \end{tikzpicture}

```



この例では `annot/.style={color=#1, draw}` でスタイルを定義しています。#1 の部分に注目してください。このとき引数を1つ取ることができ、使用時に値を指定することで、その値が #1 のところに置き換えられます。例では色を引数として取っていることになります。T_EX の `newcommand` を使ったことがある読者ならばおなじみの書式であると思います。

`annot/.default=gray` で引数の既定値を定義しています。一般には **スタイル名/.default=既定値** と記述します。

スタイルを使用している箇所を見てみましょう。`node[annot=black, right]` では黒を引数として渡しています。一方、`node[annot, right]` では引数を渡していません。したがって既定値であるグレーで描画されます。

8.3. every 型のスタイル

ノードや線などの各種要素に対して同一のスタイルを適用することができます。例えば図中のすべてのノードにわたって同一のスタイルを適用したい場合は `every node/.style={設定}` と記述します。

ノード全般のスタイル

```
1 \begin{tikzpicture}[every node/.style={circle, draw}] % ノードのスタイル
2   \node at (0,0) {$p$};
3   \node at (1,0) {$q$};
4 \end{tikzpicture}
```



この例ではすべてのノードに `circle, draw` が適用されています。適用対象となるノードはすべてなので、使用する側はスタイル名を指定する必要はありません。

もし別のスタイルを指定した場合はそちらが優先されます。

別のスタイルを適用

```
1 \begin{tikzpicture}[every node/.style={gray, draw},
2   except/.style={black}]
3   \node at (0,0) {$p$};
4   \node[except] at (1,0) {$q$};
5 \end{tikzpicture}
```

p q

この例では `every node/.style={gray, draw}` によりすべてのノードをグレーで描画するよう指定し、また `except` スタイルを黒を指定するものとして定義しています。3 行目ではオプション指定をしていないため、`every node` によりグレーで描画されます。4 行目ではスタイル `except` を指定しているため、こちらが優先され黒で描画されます。一方、`every node` の `draw` オプションキーは相変わらず有効なため、ノードの輪郭は描画されます。

`every` 型のスタイル指定はいろいろありますが、以下にその一部を掲載します。

スタイル指定	説明
<code>every picture</code>	すべての図要素。 <code>\tikzset</code> コマンドで定義すること。
<code>every path</code>	すべての <code>path</code>
<code>every circle</code>	すべての <code>circle</code>
<code>every to</code>	すべての <code>to</code> 。なお <code>--</code> には適用されない。
<code>every node</code>	すべての <code>node</code>
<code>every rectangle node</code>	形状が <code>rectangle</code> であるすべての <code>node</code>
<code>every label</code>	<code>label</code> オプションキーで作成されたすべての <code>node</code>

他にもいろいろあります。Till Tantau 氏の `pgfmanual` をご参照ください。

第 9 章 数学エンジン編

この章では、PGF の数学エンジンにかかわる事項を解説します。

なお、詳細な情報を知りたい方は、Till Tantau 氏の `pgfmanual` をご参照ください。本書ではその中から便利と筆者が思うことを中心に取り上げます。

9.1. マクロの定義

作図において点の位置を計算によって求めたい場合があります。そのとき変数のようなものを使えとたいへん便利です。T_EX にはマクロという概念がありますが、これを変数のように使うことができます。特に PGF の数学エンジンをも扱えるようにした PGF 用のマクロ定義文があります。

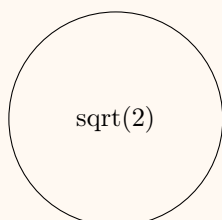
マクロの定義は `\def` を使い、`\def\マクロ名{置換後の文字列}` の書式で記述します。

マクロ定義

```

1 \begin{tikzpicture}
2   \def\r{sqrt(2)} % \r に√2 を代入
3   \draw (0,0) circle[radius=\r] node {\r};
4 \end{tikzpicture}

```



上の例ではマクロ `\r` に `sqrt(2)` を定義しています。

あたかも変数 `\r` に $\sqrt{2}$ を代入しているように見えますし、`radius=\r` により、実際に半径 $\sqrt{2} = 1.41421\dots$ の円が描画されています。しかしラベルを見てください。1.41421 ではなく `sqrt(2)` と表示されています。このことから実際には `\r` と記述された箇所が単に `sqrt(2)` と置換されたに過ぎないことが分かります。つまり

マクロ展開後

```
1 \draw (0,0) circle[radius=sqrt(2)] node {sqrt(2)};
```

と等価ということです。このようにマクロとは置換のルールを定義したものとと言えます。また置換することをマクロを展開と呼びます。

以下の場合にはエラーになります。

```
1 \begin{tikzpicture}
2   \def\r{sqrt(2)}
3   \draw (0,0) circle[radius=\r cm] node {\r};
4 \end{tikzpicture}
```

半径に単位 `cm` をつけたかったのですが、展開すると

```
1 \draw (0,0) circle[radius=sqrt(2) cm] node {sqrt(2)};
```

となり、エラーとなります。

このような場合に対処するために PGF には式の評価後の値をマクロ定義するコマンドがあります。結果が数値の場合は `\pgfmathsetmacro` を、長さ（の次元を持つ量）の場合は `\pgfmathsetlengthmacro` を使います。

数値のマクロ

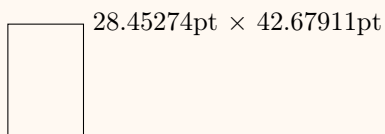
```
1 \begin{tikzpicture}
2   \pgfmathsetmacro\r{sqrt(2)} % \r に  $\sqrt{2}$  を代入
3   \draw (0,0) circle[radius=\r cm] node {\r};
4 \end{tikzpicture}
```



上の例を見ると、ラベルには `sqrt(2)` の評価後の値 1.41421 が表示されていることが分かります。

長さのマクロ

```
1 \begin{tikzpicture}
2   \pgfmathsetlengthmacro\a{1cm} % \a に 1cm を代入
3   \pgfmathsetlengthmacro\b{\a + 5mm} % \b に \a + 5mm を代入
4   \draw (0,0) rectangle (\a, \b) node[right] {\a \ $\times$ \b};
5 \end{tikzpicture}
```



上の例は `\pgfmithsetlengthmacro` を使った場合です。ラベルを見ると `pt` を単位とした長さが表示されています。

`\pgfmithsetmacro` と `\pgfmithsetlengthmacro` は式を評価してからマクロ定義をすると述べましたが、いったんマクロ定義したあとは、展開時は通常の置換ルールに従うことに注意してください。例えば次の例を見てください。

数値のマクロ2

```
1 \pgfmithsetmacro{\a}{-1}
2 \pgfmithsetmacro{\b}{\a^2}
3 \pgfmithsetmacro{\c}{(\a)^2}
4
5 $a= \a, b= \b, c= \c$
```

$a = -1.0, b = -1.0, c = 1.0$

`\b` の定義では `\a^2` → `-1^2` と展開されるため、式評価の結果 `-1` となります。一方、`\c` の定義では `(\a)^2` → `(-1)^2` と展開されるため、式評価の結果 `1` となります。

なおマクロ定義において、文末に `;` が不要であることに注意してください。マクロ定義の構文自体は `TEX` や `PGF` のものであり、`TikZ` とは無関係だからです。

同じ理由で、マクロは `tikzpicture` 環境外でも使えます。さきほどの例がちょうどそれに当たります。

`math` ライブラリを使うと、マクロ定義をもっと簡単な構文で行えます。事前に `math` ライブラリを読みこむ必要があります。

マクロ定義は `\tikzmath` コマンドで行います。

math ライブラリによるマクロ定義

```
1 %\usetikzlibrary{math}
2 \tikzmath{
3   \a=-1;
4   \b=\a^2;
5   \c=(\a)^2;
6 }
7 $a= \a, b= \b, c= \c$
```

$a = -1.0, b = -1.0, c = 1.0$

`\tikzmath` コマンドはマクロ定義だけでなく、関数の定義や繰り返し制御、条件分岐も記述できます。詳細は Till Tantau 氏の `pgfmanual` をご参照ください。

例から分かるように `tikzpicture` 環境外でも使えます。また `\tikzmath` コマンドの中では、各文末にセミコロン ; をつけなければなりません。一方、`\tikzmath` コマンド自身は ; で終える必要はありません。

9.2. 演算子と数学関数

PGF にはいくつかの演算子と数学関数が用意されています。

まずは演算子から。

演算子	意味	演算子	意味
+	和	-	差
*	積	/	商
^	べき乗	!	階乗

ほかにも計算を優先して行う () も使用できます。

次に数学関数です。

演算子	意味	演算子	意味
<code>div(x,y)</code>	x/y の整数部	<code>sqrt(x)</code>	平方根 \sqrt{x}
<code>factorial(x)</code>	階乗 $x!$	<code>pow(x,y)</code>	べき乗 x^y
<code>e</code>	自然対数の底 2.718281828	<code>exp(x)</code>	指数関数 e^x
<code>ln(x)</code>	自然対数 $\ln x$	<code>log10(x)</code>	底 10 の対数 $\log_{10} x$
<code>abs(x)</code>	絶対値 $ x $	<code>pi</code>	円周率 3.141592654
<code>rad(x)</code>	度からラジアンへ変換	<code>deg(x)</code>	ラジアンから度へ変換
<code>sin(x)</code>	正弦関数 $\sin x$	<code>cos(x)</code>	余弦関数 $\cos x$
<code>tan(x)</code>	正接関数 $\tan x$	<code>asin(x)</code>	逆正弦関数 $\sin^{-1} x$
<code>acos(x)</code>	逆余弦関数 $\cos^{-1} x$	<code>atan(x)</code>	逆正接関数 $\tan^{-1} x$
<code>min(x1,x2,...,xn)</code>	最小値	<code>max(x1,x2,...,xn)</code>	最大値
<code>veclen(x,y)</code>	$\sqrt{x^2 + y^2}$		

演算子や関数はここに掲載した以外にも多数あります。Till Tantau 氏の `pgfmanual` をご参照ください。

上の表の `x` や `y` の箇所には数値のほかマクロを指定することもできます。以下使用例を示します。

関数の使用例

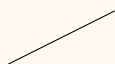
```
1 \begin{tikzpicture}
2   \pgfmathsetmacro\x{1}
3   \pgfmathsetmacro\y{2}
4   \pgfmathsetmacro\z{exp(\x) + exp(\y)}
5   \node (0,0) {$e^{\x} + e^{\y} = \z$};
6 \end{tikzpicture}
```

$$e^1 + e^2 = 10.1072$$

これらの関数はマクロ定義だけでなく、TikZ のコマンドの中でも使うことができます。

関数の使用例 2

```
1 \begin{tikzpicture}
2   \draw (0,0) -- ({2*cos(pi/4 r)}, {sin(pi/4 r)});
3 \end{tikzpicture}
```



第 10 章 制御コマンド編

TikZ には繰り返しや条件分岐など、いわゆるプログラムの世界で言うところの制御構文があります。

この章では、制御構文を解説します。

10.1. 繰り返し制御

同じコマンドを繰り返し実行したい場合は `\foreach` コマンドを使います。

なお、このコマンドを提供するのは `pgffor` パッケージであり、TikZ パッケージをロードするときに一緒にロードされます。TikZ とは別のパッケージなので `tikzpicture` 環境以外でも使用することができます。

基本

```

1 \begin{tikzpicture}
2   \coordinate (A) at (0,0); \coordinate (B) at (1,0); \coordinate (C) at (2,0);
3
4   \foreach \P in {A,B,C} \fill (\P) circle[radius=2pt]; % 点 A,B,C に黒点をつける
5 \end{tikzpicture}

```

• • •

この例では `\P` に値 `A`、`B`、`C` を順次代入しながら、その都度 `\fill` コマンドを実行しています。結果として次と等価になります。

```

1 \begin{tikzpicture}
2   \coordinate (A) at (0,0); \coordinate (B) at (1,0); \coordinate (C) at (2,0);
3
4   \fill (A) circle[radius=2pt];
5   \fill (B) circle[radius=2pt];
6   \fill (C) circle[radius=2pt];
7 \end{tikzpicture}

```

一般には `\foreach マクロ名 in {リスト} {コマンド}` の構文で記述します。

マクロ名は `\` で始めます。`\rad` など複数の文字からなる文字列も使用可能です。リストには代入したい値をカンマ区切りで記述し、それらを波カッコで囲います。そのあとに繰り返し処理したいコマンドを波カッコで囲います。コマンドは複数でも構いません。なお、コマンドが 1 つだけの場合は波カッコを省略することができます。すると、リストの要素の数だけコマンドが実行され、その際コマンド内のマクロが記述された箇所が、それらの要素に置換されます。

上の例ではマクロにノード名を代入しています。それ以外にも数値、文字列や座標などを代入することもできます。

座標を代入

```
1 \begin{tikzpicture}
2   \foreach \P in {(0,0),(1,0),(2,0)} \fill \P circle[radius=2pt];
3 \end{tikzpicture}
```

● ● ●

この例では座標 (0,0)、(1,0) および (2,0) を `\P` に代入しています。`\fill \P` としていることに注意しましょう。`\fill (\P)` としてしまうと、実行時に `\fill ((0,0))` などと置換されてしまいます。

`/` で区切ることで複数のマクロを指定することもできます。そのときはリストの各要素も `/` で区切って指定します。

2 つの要素を代入

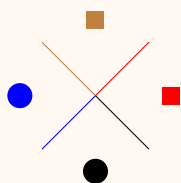
```
1 \begin{tikzpicture}
2   \foreach \angle/\str in {0/東, 180/西, 90/北, 270/南} \node at (\angle:1) {\str};
3 \end{tikzpicture}
```

北
西 東
南

上の例では `\angle` には極座標の角度、`\str` にはラベル名を代入しています。

3 つの要素を代入

```
1 \begin{tikzpicture}
2   \foreach \angle/\shp/\col in {0/rectangle/red, 180/circle/blue, 90/rectangle/brown,
3     270/circle/black}
4     {\node[fill=\col, \shp] at (\angle:1) {};}
5   \draw[\col] (0,0) -- (\angle + 45:1);}
6 \end{tikzpicture}
```



上の例では `\angle` には極座標の角度、`\shp` にはノードの形状そして `\col` には色名を代入しています。
 $(\angle + 45:1)$ は置換の結果 $(0 + 45:1)$ 、 $(180 + 45:1)$ などとなります。

マクロの置換はまさにそのままの置換なので、次の例のようにノードへの命名などにも使えます。

```
1 \foreach \name/\pos in {A/0, B/1, C/2} \coordinate (\name) at (\pos,0);
```

要素のリストの列挙では省略形が使えます。

$\{1, 2, \dots, 10\}$ は $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ と解釈されます。

$\{1, 3, \dots, 10\}$ は $\{1, 3, 5, 7, 9\}$ と解釈されます。

$\{10, 9, \dots, 1\}$ は $\{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$ と解釈されます。

$\{p, \dots, s\}$ は $\{p, q, r, s\}$ と解釈されます。

冒頭でも述べたように `foreach` コマンドは `tikzpicture` 環境以外でも使えます。

tikzpicture 環境外

```
1 \foreach \n in {0, ..., 10} {$2^{\n}, \ $}
```

$2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10},$

最後がセミコロン ; で終わっていないことに注意しましょう。`foreach` は `pgffor` パッケージのコマンドなので、`TeX` コマンドと同様、文末をセミコロンで終える必要はありません。今までの例でセミコロンで終わっていたのは、それらが `TikZ` のコマンドだったためです。

第 11 章 図の配置編

この章では、図の配置を指定する方法を解説します。

11.1. 図の描画領域を取得する

図を配置するにあたって、その描画領域の情報を取得できると便利です。

描画領域は `current bounding box` で取得します。実はこれ自体 1 つのノードです。

基本

```

1 \begin{tikzpicture}
2   \path (0,0) rectangle (3,2);
3
4   \fill[lightgray] (current bounding box.south west) rectangle (current bounding box.north
5     east);
6   \node[below right] at (current bounding box.north west) {左上};
7   \node[above left] at (current bounding box.south east) {右下};
8 \end{tikzpicture}

```

左上

右下

上の例ではまず (0,0) と (3,2) を対角とする長方形を用意しています (`\path` コマンドを使っているので線は描画されませんが)。この時点でこの長方形が描画領域全体となります。4 行目では描画領域をグレーで塗りつぶしています。対角となる点を `current bounding box.south west` と `current bounding box.north east` で指定しています。このように `current bounding box` 自体はノードです。5 行目と 6 行目で描画領域の隅にラベルを配置しています。

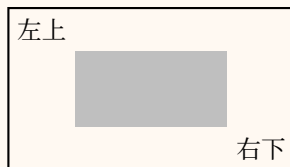
`current bounding box` はあくまでも「現在」の描画領域を表します。図要素を追加していくと、そのたびに描画領域は広がっていきます。

現在の描画領域

```

1  \begin{tikzpicture}
2      \path (0,0) rectangle (2,1);
3
4      \fill[lightgray] (current bounding box.south west) rectangle (current bounding box.north
5      east);
6      \node[above left] at (current bounding box.north west) {左上};
7      \node[below right] at (current bounding box.south east) {右下};
8
9      \draw[thick] (current bounding box.south west) rectangle (current bounding box.north
10     east);
11 \end{tikzpicture}

```



上の例の 2 行目の時点では、描画領域は (0,0) と (2,1) を対角とする長方形の領域です。ですので、グレーで塗りつぶされる領域もその長方形に内部になります。そのあと 5 行目と 6 行目でラベルを 2 つ配置したことで描画領域が広がります。実際、8 行目で長方形を描画すると、ラベルを含む領域を囲むように枠線が描画されています。

11.2. 図全体を枠線で囲む

`backgrounds` ライブラリを使うことで、図全体を枠線で囲むことができます。事前に `backgrounds` ライブラリを読みこむ必要があります。

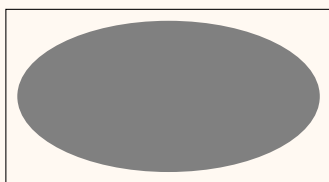
`tikzpicture` 環境のオプションとして `show background rectangle` スタイルを指定すると背景を描画します。

基本

```

1  %\usetikzlibrary {backgrounds}
2  \begin{tikzpicture}[show background rectangle]
3      \fill[gray] (0,0) circle[x radius=2, y radius=1];
4  \end{tikzpicture}

```



枠線の色は、`background rectangle` スタイルの定義を通して指定します。図の描画領域と枠線との間に間隔を取

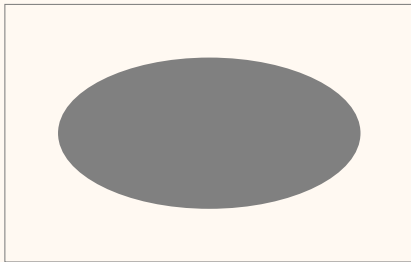
りたい場合は `inner frame sep` オプションキーを指定します。

枠線のオプション

```

1 \usetikzlibrary {backgrounds}
2 \begin{tikzpicture}[background rectangle/.style={draw=gray},
3                     inner frame sep=20pt,
4                     show background rectangle]
5   \fill[gray] (0,0) circle[x radius=2, y radius=1];
6 \end{tikzpicture}

```



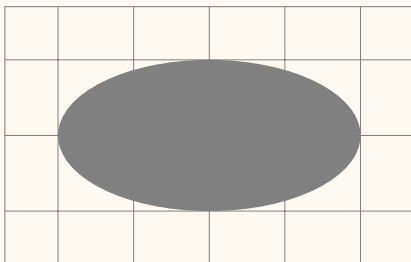
`show background grid` を指定すると背景にグリッド線が描画されます。

背景にグリッド線を描画

```

1 \usetikzlibrary {backgrounds}
2 \begin{tikzpicture}[background rectangle/.style={draw=gray},
3                     inner frame sep=20pt,
4                     show background rectangle,
5                     show background grid]
6   \fill[gray] (0,0) circle[x radius=2, y radius=1];
7 \end{tikzpicture}

```



図全体に背景色をつけることもできます。4.8 節を参照してください。

11.3. 図を並べる

複数の図を並べるには大きく分けて、次の 2 つがあります。すなわち、複数の `tikzpicture` 環境で作成した図を並べる方法と、1 つの `tikzpicture` 環境の中に複数の図を並べる方法です。それぞれの図にキャプションを入れたい場合は前者の方法をとります。

11.3.1. figure 環境を使う

複数の `tikzpicture` 環境で作成した図を配置するときは `figure` 環境の中に `minipage` 環境を入れ子にします。なお、`figure` 環境も `minipage` 環境も TikZ コマンドではありませんので、これらは TikZ 以外の一般の図を貼り付けるときでも使用できます。

例えば次のように記述すると 2 つの図が横に並んで配置されます。

2 つの図を横に並べる

```

1 \begin{figure}
2   \begin{minipage}[位置]{横幅の長さ}
3     ...
4   \end{minipage}
5   \caption{小キャプション}
6   %
7   \begin{minipage}[位置]{横幅の長さ}
8     ...
9   \end{minipage}
10  \caption{小キャプション}
11 \end{figure}
12 \caption{キャプション}

```

例を見てみましょう。

2 つの図を横に並べる

```

1 \begin{figure}[H]
2   \centering
3   \begin{minipage}[b]{0.3\columnwidth}
4     \centering
5     \begin{tikzpicture}
6       \draw (0,0) rectangle (2,2);
7       \draw[fill=white] (1,1) circle[radius=1];
8       \draw[fill=white] (0,0) -- (1.5,0) -- (60:1.5) -- cycle;
9     \end{tikzpicture}
10    \caption*{例 1}
11  \end{minipage}
12  %
13  \begin{minipage}[b]{0.3\columnwidth}
14    \centering
15    \begin{tikzpicture}

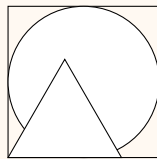
```



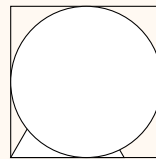
```

6      \draw (0,0) rectangle (2,2);
7      \draw[fill=white] (0,0) -- (1.5,0) -- (60:1.5) -- cycle;
8      \draw[fill=white] (1,1) circle[radius=1];
9      \end{tikzpicture}
10     \caption*{例 2}
11 \end{minipage}
12 \caption{複数の図を並べる例}
13 \end{figure}

```



例 1



例 2

図 11.1: 複数の図を並べる例

位置のところには、次のオプション値を指定できます。

オプション値	説明
t	ミニページの最上行のベースラインを本文のベースラインに合わせる
c	ミニページの中央を本文のベースラインに合わせる
b	ミニページの最下行のベースラインを本文のベースラインに合わせる

位置のオプション値を省略したときのデフォルトは c です。

それぞれの違いをみてください。

b を指定

```

1  \begin{minipage}[b]{0.3\columnwidth}
2      \centering
3      \begin{tikzpicture}
4          \draw (0,0) rectangle (2,2);
5      \end{tikzpicture}
6  \end{minipage}
7  %
8  本文
9  %
10 \begin{minipage}[b]{0.3\columnwidth}
11     \centering
12     \begin{tikzpicture}
13         \draw (0,0) rectangle (2,1);

```

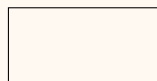
```

4 \end{tikzpicture}
5 \end{minipage}

```



本文



c を指定

```

1 \begin{minipage}[c]{0.3\columnwidth}
2 \centering
3 \begin{tikzpicture}
4 \draw (0,0) rectangle (2,2);
5 \end{tikzpicture}
6 \end{minipage}
7 %
8 本文
9 %
10 \begin{minipage}[c]{0.3\columnwidth}
11 \centering
12 \begin{tikzpicture}
13 \draw (0,0) rectangle (2,1);
14 \end{tikzpicture}
15 \end{minipage}

```



本文



t を指定

```

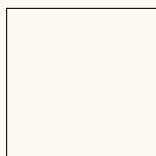
1 \begin{minipage}[t]{0.3\columnwidth}
2 \centering
3 \begin{tikzpicture}
4 \draw (0,0) rectangle (2,2);
5 \end{tikzpicture}
6 \end{minipage}
7 %
8 本文
9 %
10 \begin{minipage}[t]{0.3\columnwidth}
11 \centering
12 \begin{tikzpicture}

```

```

3      \draw (0,0) rectangle (2,1);
4      \end{tikzpicture}
5  \end{minipage}

```



本文



例を見ると位置オプション値が `b` と `c` の場合は、想定通りの結果が出ていると思います。ところが `t` の場合は図の上部に「本文」の文字列が来るのではなく、むしろ下部に来ています。まるで `b` を指定したときと同じ挙動に見えます。

実は、ミニページの中に図が 1 つだけある状態は、あたかも文が 1 行だけあるのと同じになります。したがって、ミニページの最上行と最下行は同じ行にあたります。ベースラインは図の下部にありますので、「本文」の位置は結局 `b` を指定したときと同じになるわけです。

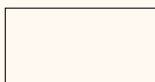
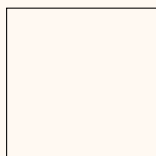
左右の図で上辺の位置を合わせるには、以下のように図のベースラインを変更します。

ベースラインを変更

```

1  \begin{minipage}[t]{0.3\columnwidth}
2      \centering
3      \begin{tikzpicture}[baseline=(current bounding box.north)]
4          \draw (0,0) rectangle (2,2);
5      \end{tikzpicture}
6  \end{minipage}
7  %
8  \begin{minipage}[t]{0.3\columnwidth}
9      \centering
10     \begin{tikzpicture}[baseline=(current bounding box.north)]
11         \draw (0,0) rectangle (2,1);
12     \end{tikzpicture}
13 \end{minipage}

```



上の例では、ベースラインを図の描画領域の上端にするよう指定しています。つまり、`tikzpicture` 環境の `baseline` オプションキーに、`current bounding box.north` アンカーを指定しています。`current bounding box` は描画領域を表します。`current bounding box` は自動的に生成されるノードであり、その位置と大きさは描画領域と一致するように生成されます。

位置オプションを指定するときは、ミニページの右と左で異なる値を指定してももちろん構いません。

ところで、複数の `minipage` 環境を並べるときに、前の `\end{minipage}` と次の `\begin{minipage}` の間に 1 行

空けてしまうと、図が横ではなく縦に並んでしまいます*1。一方、`\end{minipage}` と `\begin{minipage}` の行間を詰めてしまうと、コードが見にくくなってしまいます。そこで、

```

1   ...
2
3   \end{minipage}
4   %
5   \begin{minipage}
6   ...

```

のようにコメントアウトしてしまえば、空の行とみなされず、結果的に横に並ぶようになります。あるいは、

```

1   ...
2
3   \end{minipage}
4   \qqquad
5   \begin{minipage}
6   ...

```

のようにすれば、図の間隔を確保することもできますね。

11.3.2. マトリックス状に並べる

図をマトリックス状に並べる場合は、`figure` 環境の中に `tabular` 環境を入れ、その中にミニページを入れています。

マトリックス状に並べる

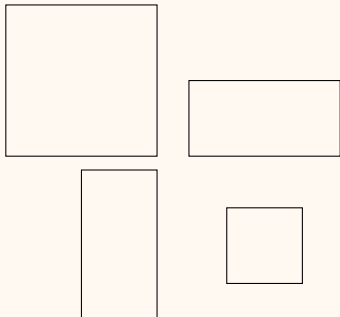
```

1   \begin{tabular}{rc}
2       \begin{minipage}[b]{2cm} \centering
3           \tikz \draw (0,0) rectangle (2,2);% 左上
4       \end{minipage}
5       &
6       \begin{minipage}[b]{2cm} \centering
7           \tikz \draw (0,0) rectangle (2,1);% 右上
8       \end{minipage}
9       \\
10      \begin{minipage}[c]{1cm} \centering
11          \tikz \draw (0,0) rectangle (1,2);% 左下
12      \end{minipage}
13      &
14      \begin{minipage}[c]{1cm} \centering
15          \tikz \draw (0,0) rectangle (1,1);% 右下
16      \end{minipage}

```

*1 おそらく改行とみなされるからではないかと思います。

```
7 \end{tabular}
```



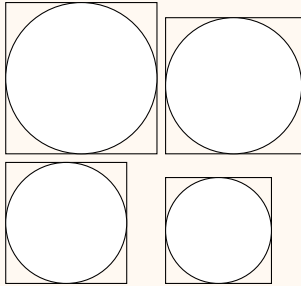
横同士の図の位置を合わせるには、前述のとおり `minipage` 環境のオプションを使用します。一方、縦同士の図の位置を合わせるには、`tabular` 環境の列配置の指定を行います。上の例では 1 列目を右寄せ `r`、2 列目をセンタリング `c` にしています。

11.3.3. tikzpicture 環境に複数並べる

1 つの `tikzpicture` 環境の中に複数の図を並べるには `\matrix` コマンド (3.4 節参照) を使います。

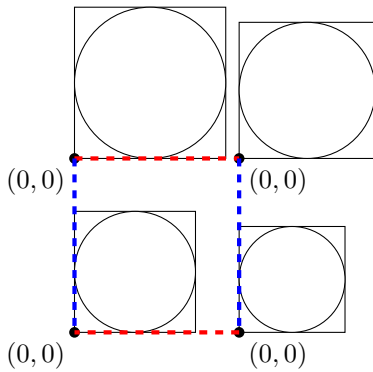
マトリックス状に配置

```
1 \begin{tikzpicture}
2   \matrix [row sep=0.1cm, column sep=0.1cm] {
3     \draw (0,0) rectangle (2,2);
4     \draw[fill=white] (1,1) circle[radius=1];
5     &
6     \draw (0,0) rectangle (1.8,1.8);
7     \draw[fill=white] (0.9,0.9) circle[radius=0.9];
8     \\
9     \draw (0,0) rectangle (1.6,1.6);
10    \draw[fill=white] (0.8,0.8) circle[radius=0.8];
11    &
12    \draw (0,0) rectangle (1.4,1.4);
13    \draw[fill=white] (0.7,0.7) circle[radius=0.7];
14    \\
15    };
16 \end{tikzpicture}
```



上の例ではマトリックスの各セルに図形を描画しています。このとき図形間の相対的な位置は、各図の原点 $(0,0)$ がそろるように位置合わせされます。

図形間の相対的な位置



`\matrix` コマンドを使うときは、最後の行も `\\` で終わることを忘れないようにしましょう。

図の位置合わせを任意に行いたい場合は、いったん図形を 1 つのノードの中に入れてしまい、ノードに対して位置合わせの指定を行います。例えばそれぞれの図を各セルの中央に配置したい場合は、以下のようにします。

各セルの中央に配置

```

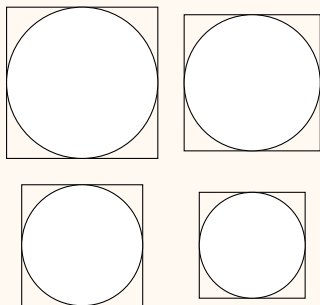
1 \begin{tikzpicture}
2   \matrix [row sep=0.1cm, column sep=0.1cm,
3     every cell/.style={anchor=center}]
4   {
5     \node{\tikz{
6       \draw (0,0) rectangle (2,2);
7       \draw[fill=white] (1,1) circle[radius=1];}}
8   };
9   &
10  \node{\tikz{
11    \draw (0,0) rectangle (1.8,1.8);
12    \draw[fill=white] (0.9,0.9) circle[radius=0.9];}}
13  };

```

```

4      \\\
5      \node{\tikz{
6          \draw (-0.8,-0.8) rectangle (0.8,0.8);
7          \draw[fill=white] (0,0) circle[radius=0.8];}
8      };
9      &
10     \node{\tikz{
11         \draw (0,0) rectangle (1.4,1.4);
12         \draw[fill=white] (0.7,0.7) circle[radius=0.7];}
13     };
14     \\\
15     };
16 \end{tikzpicture}

```



上の例では、各セルに配置する図を `\node` コマンドの波カッコ `{ }` の中に記述しています。これで図形を中に持つノードが作成されます。そして、`\matrix` コマンドのオプションとして `every cell/.style={anchor=center}` を指定しています。これですべてのセルのノードに対して `anchor=center` を指定したことになります（ここでは説明のために `every cell/.style={anchor=center}` を明示的に記述しましたが、そもそもデフォルトが `center` なので、中央寄せの場合はこのコードは実は省略できます）。

それぞれのセルで異なる位置合わせを行う場合は、各 `\node` コマンドごとにオプションを指定するのが早いでしょう。

セルごとに位置合わせ

```

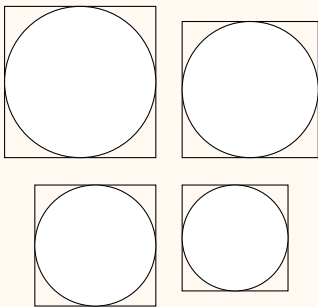
1 \begin{tikzpicture}
2   \matrix [row sep=0.1cm, column sep=0.1cm]
3   {
4     \node[anchor=south east]{\tikz{
5       \draw (0,0) rectangle (2,2);
6       \draw[fill=white] (1,1) circle[radius=1];}
7     };
8     &
9     \node[anchor=south west]{\tikz{
10      \draw (0,0) rectangle (1.8,1.8);
11      \draw[fill=white] (0.9,0.9) circle[radius=0.9];}

```

```

2      };
3      \\
4      \node[anchor=north east]{\tikz{
5          \draw (0,0) rectangle (1.6,1.6);
6          \draw[fill=white] (0.8,0.8) circle[radius=0.8];}
7      };
8      &
9      \node[anchor=north west]{\tikz{
10         \draw (0,0) rectangle (1.4,1.4);
11         \draw[fill=white] (0.7,0.7) circle[radius=0.7];}
12     };
13     \\
14     };
15     \end{tikzpicture}

```



上の例では左上の図を右下に寄せ、右上の図を左下に寄せ、左下の図を右上に寄せ、そして右下の図を左上に寄せています。

マトリックス状にこだわらず、任意の位置に図を配置したいときは、それぞれの図を `scope` 環境で囲み、`shift` オプションキーで移動量を指定すると作図しやすいと思います。

shift オプションで調整

```

1 \begin{tikzpicture}
2   \begin{scope}
3     \draw (0,0) rectangle (2,2);
4     \draw[fill=white] (1,1) circle[radius=1];
5   \end{scope}
6   \begin{scope}[shift={(2.1cm,0.2cm)}]
7     \draw (0,0) rectangle (1.8,1.8);
8     \draw[fill=white] (0.9,0.9) circle[radius=0.9];
9   \end{scope}
10  \begin{scope}[shift={(1cm,-1.7cm)}]
11    \draw (0,0) rectangle (1.6,1.6);
12    \draw[fill=white] (0.8,0.8) circle[radius=0.8];
13  \end{scope}

```



```
4 \begin{scope}[shift={(3.1cm,-1.3cm)}]  
5   \draw (0,0) rectangle (1.4,1.4);  
6   \draw[fill=white] (0.7,0.7) circle[radius=0.7];  
7 \end{scope}  
8 \end{tikzpicture}
```

