TikZ への入門

2022 年 11 月 23 日

目次

第 1 章　TikZ の基礎

1.1.　作図の要素

この章では、TikZ を使って図を作成するときの要素を解説します。この章の執筆にあたっては、Till Tantau 氏の pgfmanual[*1] の内容を参考にさせていただきました。

Inkscape などのグラフィックソフトとは違い、TikZ ではコマンドを記述しながら作図を行います。その意味では、TEX でコマンドを記述しながら文書を作成するのとたいへん似ています。

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. 座標を指定する構文
2. パスを指定する構文
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

それではこれらを少し詳しく見ていきましょう。

1.1.1.　座標を指定する構文

座標を指定するにはいくつかの構文があります。

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方法で、(x 座標,y 座標) の形式になります。数値の間はコンマで区切ります。

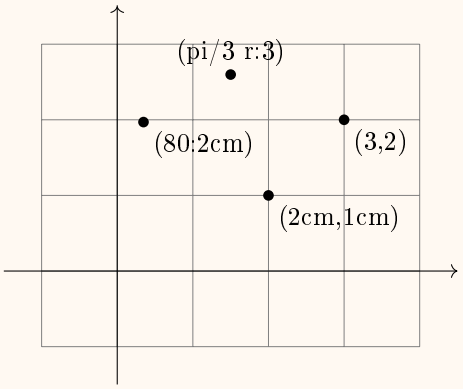
また極座標を使って (偏角:動径長さ) と指定することもできます。このときはコンマではなくコロンで区切ることに注意してください。

三次元の図を作成するときは、(x 座標,y 座標,z 座標) と指定することができます。

[*1] https://www.ctan.org/pkg/pgf

基本

```
\begin{tikzpicture}
    \draw[help lines] (-1,-1) grid (4,3);
    \draw[->] (-1.5,0) |- (4.5,0);
    \draw[->] (0,-1.5) |- (0,3.5);

    \fill (2cm,1cm) circle[radius=2pt] node[below right] {(2cm,1cm)};% 点 (2cm,1cm)
    \fill (80:2cm) circle[radius=2pt] node[below right] {(80:2cm)};% 点 (45:2cm)
    \fill (3,2) circle[radius=2pt] node[below right] {(3,2)};% 点 (3cm,2cm)
    \fill (pi/3 r:3) circle[radius=2pt] node[above] {(pi/3 r:3)};% 点 (60:3cm)
\end{tikzpicture}
```



上の例の 7 行目を見てみましょう。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。

8 行目では角度の指定をラジアンで行っています。最後に 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。

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。詳細は本書リファレンスを参照してください。

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。例えばカレントポジションが (1cm, 2cm) だとすると、++(2cm,3cm) と記述することで (3cm,5cm) という座標になります。同時にこの座標が新しいカレントポジションになります。一方、|+(2cm,3cm) と記述すると、やはり (3cm,5cm) という座標になりますが、このときはカレントポジションは変更されません。つまり ++ と + の違いはカレントポジションを変更するかしないかの違いになります。

1.1.2.　パスを指定する構文

TikZ での構文は以下の形をとります。

\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。

コマンドは具体的には \path などを指します。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。

描画の指定を記述したものはオペレーションと呼びます。座標指定やオペレーションは連続して記

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]

(0pt,0pt) -- (1pt,0pt) (2pt,0pt) -- (3pt,0pt) はオペレーション列の例です。また、その結果としての線分の列 .. がパスとなります。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のことをアクションと呼びます。

さて、TikZ で図を描くことは、それはパスを順次作成することを意味します。図形上連結していなくても（例えば線で結ばれていなくても）パスとしては連続しているとみなします。例えば (0pt,0pt) -- (1pt,0pt) (2pt,0pt) -- (3pt,0pt) は図形 .. としては 2 本に分かれた線分になっていますが、パスとしては 1 つと考えます。

例えば三角形のパスを指定するなら、オペレーション列を

```
(
5pt,0pt) -- (0pt,0pt) -- (0pt,5pt) -- cycle
```

とすれば、◺ というパスが描かれます。

1.1.3.　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、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という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。もちろんこれだけだと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（ 1.1.4 節を参照）。

[*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**は本書独特の用語です。同マニュアルを見ましたが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。

描画のアクション

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

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

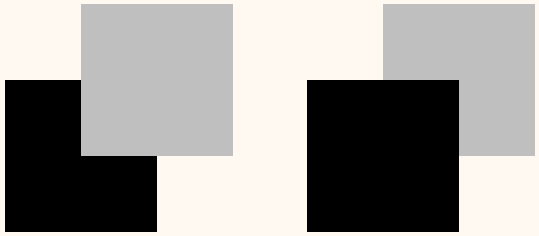
と記述することもできます。

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 つの図形が重なる場合は後で記述した方のコマンドが上位に来ます。

重なる図形

```
\begin{tikzpicture}
    \begin{scope}
        \fill[black] (0,0) rectangle (2,2);
        \fill[lightgray] (1,1) rectangle (3,3);
    \end{scope}
    \begin{scope}[xshift=4cm]
        \fill[lightgray] (1,1) rectangle (3,3);
        \fill[black] (0,0) rectangle (2,2);
    \end{scope}
\end{tikzpicture}
```



左の例では、黒で塗りつぶした正方形を先に、グレーで塗りつぶした正方形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描画されています。右の例はその逆ですね。

1.1.4.　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

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

この例では線の色を `gray`、線の幅を `3pt` として線を描画し、塗りつぶしの色を `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`、`arrows`、`shape` 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

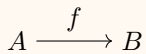
```
\begin{tikzpicture}
    \draw[gray] (1,0) -- (0,0);
    \draw[color=gray] (1,1) -- (0,1);
\end{tikzpicture}
```

1.1.5.　ノード

TikZ にはノードというものがあり、長方形、円やラベルを簡単に図に追加することができます。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。

ノード

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



上の例では ラベルを表示するためのノードを 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。テキストを空に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

```
\begin{tikzpicture}
    \draw (0,0) node[draw, circle] {$A$} (1,0) node[draw, rectangle, gray] {$B$}
    (2,0) node[draw, circle, fill=gray] {};
\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文字色を、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できます。

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。

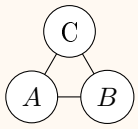
先ほどの例ではラベル A とラベル B はカレントポジションに配置されています。`(0,0) node[left] {$A$}` を見ると、`(0,0)` の時点でのカレントポジションが $(0,0)$ ですのでその位置にラベル A が配置されます。ただしオプション `left` も指定されていますので、実際には $(0,0)$ の少し左に配置されています。`(`**座標**`) node` のように座標の直後に `node` を記述します。あるいは `node at (`**座標**`)` のように `at` の後に座標を明示的に指定することもできます。`\path node at` の短縮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

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

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

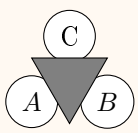


この例を見ると先に 3 つの線分が描画されてから、その上にノードが描画されることが分かります（重なりが分かるように、すべてのノードを白で塗りつぶしています）。

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

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



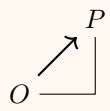
この例では、グレーで塗りつぶされた三角形を描画するコマンドが後に記述されていますので、ノードの上に重なるように描画されます。

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

```
\begin{tikzpicture}
    % ノードへの命名
    \node (O) at (0,0) {$O$};
    \coordinate (A) at (1,0);
    \node[name=P] at (1,1) {$P$};

    % ノードの参照
    \draw[->, thick] (O) -- (P);
    \draw (O) -- (A) -- (P);
\end{tikzpicture}
```



ノード名をつけるメリットは、同じ座標値を毎回書かなくてよいことにあります。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もあります。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。

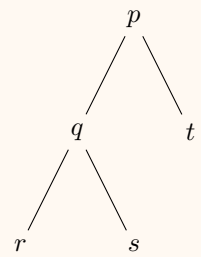
1.1.6.　ツリー、グラフのための構文

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

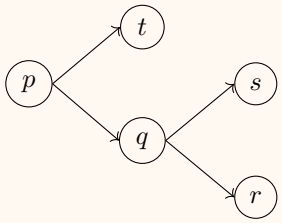
```
\begin{tikzpicture}
    \node {$p$}
        child {node {$q$}
            child {node {$r$}}
            child {node {$s$}}
        }
        child {node {$t$}
        };
\end{tikzpicture}
```



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

```
\begin{tikzpicture}
  [parent anchor=east,child anchor=west,grow=east,
   sibling distance=15mm, level distance=15mm,
   every node/.style={draw, circle},
   edge from parent/.style={draw, ->}]
    \node {$p$}
        child {node {$q$}
            child {node {$r$}}
            child {node {$s$}}
        }
        child {node {$t$}
        };
\end{tikzpicture}
```

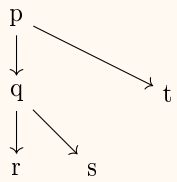


node コマンドを使うとノードの配置位置や見た目などを細かく制御できます。しかし、グラフを描くときなど画一的なノードを配置する場合は、graphs ライブラリを使うことでもっと簡潔に記述

することができます。

グラフ

```
\begin{tikzpicture}
\graph[branch right, grow down] {
  p -> { q -> {r, s}, t}
};
\end{tikzpicture}
```



1.1.7.　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

```
\begin{tikzpicture}
    \begin{scope}[thin]
        \draw (0,0) -- (2,0);
        \draw (0,-.5) -- (2,-.5);
    \end{scope}
    \begin{scope}[ultra thick, xshift=3cm]
        \draw (0,0) -- (2,0);
        \draw (0,-.5) -- (2,-.5);
    \end{scope}
\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。これらの線分一つ一つに thin を指定しているわけではないことに注意しましょう。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同じ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。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。

第 2 章　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。

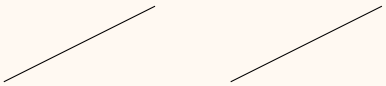
2.1.　線・図形編

この節では、直線や曲線、円や長方形の描き方を解説します。

2.1.1.　直線を描く

基本

```
\begin{tikzpicture}
    \draw (0,0) -- (2,1);% 点 (0,0) と (2,1) の間を直線で結ぶ。
    \draw (3,0) to (5,1);% 点 (3,0) と (5,1) の間を直線で結ぶ。
\end{tikzpicture}
```

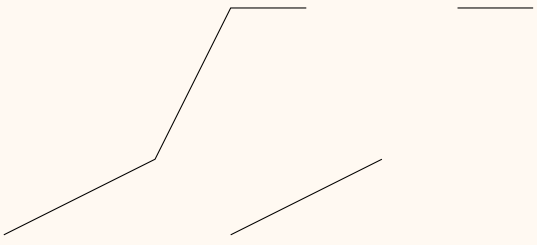


直線（正確には線分）を描くには \draw コマンド内で座標を 2 つ指定し、それらの間を -- または to でつなぎます。両者の違いは to では線分へのオプションを指定できる一方、 -- ではできません。実は to は直線だけではなく曲線を描くときにも使い、その際にオプションを指定することになります。

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だけ線分を描画しないということもできます。

線分を連続的に描画する

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

    % 点 (3,0)(5,1) 間および (6,3)(7,3) 間を直線で結ぶ。
    \draw (3,0) -- (5,1)    (6,3) -- (7,3);
\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

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

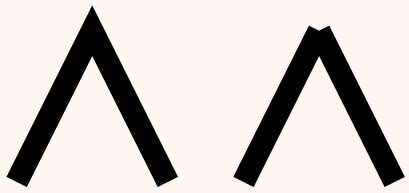
```
\begin{tikzpicture}[line width=3mm]
    \draw (0,0) -- (1,0) -- (1,1) -- cycle; %cycle で閉じる。
    \draw (2,0) -- (3,0) -- (3,1) -- (2,0); %始点と同じ点を指定して閉じる。
\end{tikzpicture}
```

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 つの線分を連続的に書く場合と別々に書く場合の違い

```
\begin{tikzpicture}[line width=3mm]
    \draw (0,0) -- (1,2) -- (2,0); %連続的に書く場合
    \draw (3,0) -- (4,2); \draw (4,2)-- (5,0); %別々に書く場合
\end{tikzpicture}
```



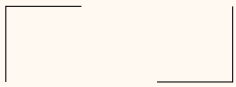
左では結合部分がきれいに接続しているのに対して、右はそうなっていません。線分を連続して書くことで、TikZ は結合部分までうまく処理してくれることが分かります。

2 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 -- の代わりに |- ま

たは -| を指定します。前者は始点から鉛直線を引き、終点へ水平線を引きます。後者は始点から水平線を引き、終点へ鉛直線を引きます。

2 点間を水平線と鉛直線の 2 本で結ぶ

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



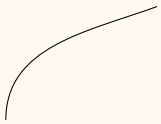
2.1.2.　曲線を描く

ここでは一般の曲線を描く方法を述べます。

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の角度（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

```
\begin{tikzpicture}
    \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
\end{tikzpicture}
```

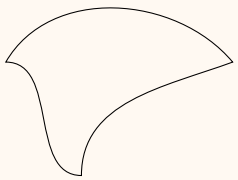


線をつなぐための to オペレーションに、out オプションで出射角を、in オプションで入射角を指定します。上の例では曲線が始点から 90 度方向つまり鉛直上方へ伸びていき、終点へは 200 度方向から到達しています。

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

```
\begin{tikzpicture}
    \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;
\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

```
\begin{tikzpicture}
\draw (0,0) to[out=0,in=-90,looseness=0.5] (1,1);
\draw[yshift=-1.5cm] (0,0) to[out=0,in=-90] (1,1);
\draw[yshift=-3cm] (0,0) to[out=0,in=-90,looseness=2] (1,1);
\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を起点としたいこともあるでしょう。その場合は relative オプションを指定します。

角度の相対指定

```
\begin{tikzpicture}
    \draw[very thick] (0,0) to[out=0,in=-90,relative] (1,1);% 角度を絶対指定
    \draw (0,0) -- (1,1);
    \draw[very thick, yshift=-1.5cm] (0,0) to[out=0,in=-90] (1,1);% 角度を相対指定
    \draw[yshift=-1.5cm] (0,0) -- (1,1);
\end{tikzpicture}
```

上が relative を指定した方の例になります。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。

これと似た指定の仕方として bend right オプションと bend left オプションを使う方法もあります。

角度の相対指定 2

```
\begin{tikzpicture}
    \draw[very thick] (0,0) to[bend right=60] (1,1);% 角度を絶対指定
    \draw (0,0) -- (1,1);
    \draw[very thick, yshift=-1.5cm] (0,0) to[bend left=60] (1,1);% 角度を相対指定
    \draw[yshift=-1.5cm] (0,0) -- (1,1);
\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− (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` $(180-(-60)=240)$ 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` オプションの他に `distance` オプションで指定できます。

膨らみの量を指定

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

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

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

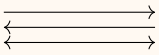
この例は、始点が (8,1)、終点が (9,4) で、制御点がそれぞれ (10,2) と (10,3.5) である 3 次ベジエ曲線になります。and の後の制御点の記述を省略すると、2 つの制御点が一致したベジエ曲線になります。

2.1.3.　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

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

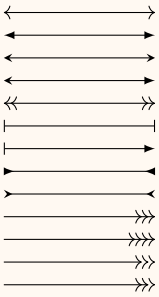


終点に矢印を追加する場合は -> を指定します。始点への追加は <-、両端への追加は <-> です。

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

矢印の形状

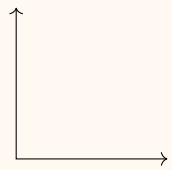
```
\begin{tikzpicture}
    \draw[<->] (0,0) -- (2,0);% 矢印
    \draw[latex-latex] (0,-0.3) -- (2,-0.3);% LaTeX 風
    \draw[stealth-stealth] (0,-0.6) -- (2,-0.6);% ステルス機風
    \draw[stealth-latex] (0,-0.9) -- (2,-0.9);% 始点と終点で異なる形状ももちろん可
    \draw[<<->>] (0,-1.2) -- (2,-1.2);% 二重矢印
    \draw[|-|] (0,-1.5) -- (2,-1.5);% 縦棒
    \draw[|-latex] (0,-1.8) -- (2,-1.8);% 写像風
    \draw[latex reversed-latex reversed] (0,-2.1) -- (2,-2.1);% 逆向き LaTeX 風
    \draw[stealth reversed-stealth reversed] (0,-2.4) -- (2,-2.4);% 逆ステルス機風
    \draw[->>>] (0,-2.7) -- (2,-2.7); %三重矢印
    \draw[->>>>] (0,-3) -- (2,-3); %四重矢印
    \draw[->.>>] (0,-3.3) -- (2,-3.3); % . を指定すると矢印部分の線分を描画しない
    \draw[->>.>] (0,-3.6) -- (2,-3.6); % . を指定すると矢印部分の線分を描画しない
\end{tikzpicture}
```



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

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

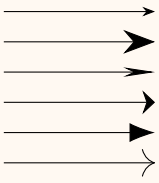


これを使うことで座標軸を描くこともできますね。

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

```
\begin{tikzpicture}
    %\usetikzlibrary {arrows.meta}
    \draw[-{Stealth}] (0,0) -- (2,0);
    \draw[-{Stealth[scale=2.5]}] (0,-0.4) -- (2,-0.4);
    \draw[-{Stealth[scale length=2.5]}] (0,-0.8) -- (2,-0.8);
    \draw[-{Stealth[scale width=2.5]}] (0,-1.2) -- (2,-1.2);
    \draw[-{Latex[scale=2]}] (0,-1.6) -- (2,-1.6);
    \draw[-{>[scale=2]}] (0,-2) -- (2,-2);
\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。

2.1.4.　線の幅を指定する

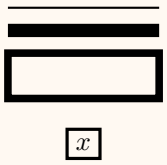
線の幅はオプションとして指定します。よく使われるものは以下の通りです。

オプション	太さ	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	

特に指定しない場合は thin が既定値になります。線の幅を直接指定したい場合は line width オプションを使います。

線の太さ

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



2.1.5.　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。以下のものがあります。

オプション	出力例
solid	———
dashed	- - - - - - - - - -
dotted	.....
dash dot	-.-.-.-.-.-.-.-
dash dot dot	-..-..-..-..-..-
double	═══
loosely dashed	- - - - - - - -
densely dashed	----------
loosely dotted	. . . . .
densely dotted	.....

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

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

二重線については、線の間隔や線の間の色を指定することができます。

二重線のオプション

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

2.1.6.　長方形を描く

基本

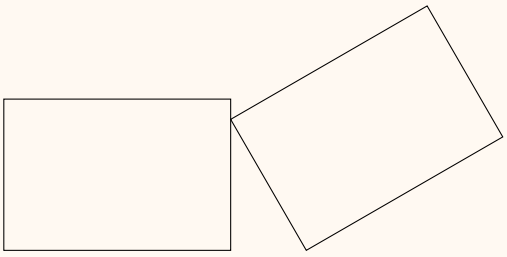
```
\begin{tikzpicture}
    \draw (0,0) rectangle (3,2);
\end{tikzpicture}
```

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かれます。

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

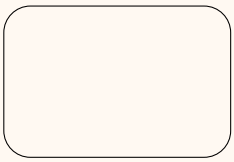
```
\begin{tikzpicture}
    \draw (0,0) rectangle (3,2);% 傾ける前の長方形
    \draw[rotate around={30:(4,0)}] (4,0) rectangle (3+4,2);% (4,0) を中心に 30 度回転した長方形
\end{tikzpicture}
```



四隅を丸めたい場合は rounded corners オプションで円の半径を指定します。

四隅が丸い長方形

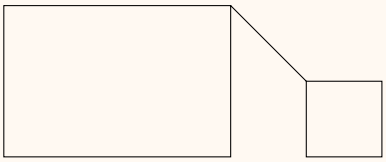
```
\begin{tikzpicture}
    \draw[rounded corners=10pt] (0,0) rectangle (3,2);% 半径 10pt の円で四隅を丸める
\end{tikzpicture}
```



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

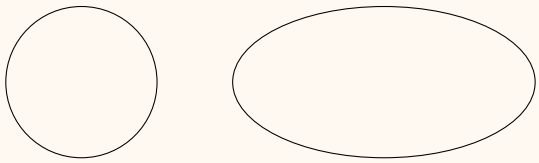
```
\begin{tikzpicture}
    \draw (0,0) rectangle (3,2) -- (4,1) rectangle (5,0);
\end{tikzpicture}
```



2.1.7.　円・楕円を描く

基本

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

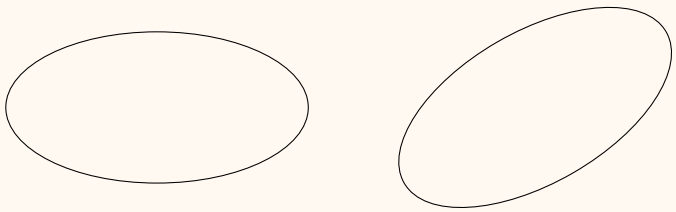


円を描くには \draw コマンド内で座標を 1 つ指定し、circle オペレーションで半径をオプション指定します。すると、指定した 1 点を中心とする円や楕円が描かれます。x radius オプションで水平方向の半径、y radius オプションで鉛直方向の半径を指定します。radius オプションを指定すると、x radius と y radius に同じ値が入り、結果として円が描かれることになります。

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

```
\begin{tikzpicture}
    \draw (0,0) circle[x radius=2cm, y radius=1cm];% 傾ける前の楕円
    \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
\end{tikzpicture}
```



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

2.1.8.　円弧を描く

基本

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

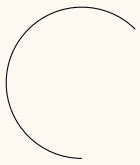
円弧を描くには \draw コマンド内で座標を 1 つ指定し、arc オペレーションで各種のオプションを指定します。すると、指定した 1 点を始点とする円弧が描かれます。指定する座標は円弧の中心ではなく、あくまでも始点であることに注意してください。上の例では点 (0,0) および (4,0) から始まる円弧がそれぞれ描かれます。

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

start angle オプションで開始角を、end angle オプション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かれます。終了角を指定する代わりに、delta angle 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

```
\begin{tikzpicture}
    \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
\end{tikzpicture}
```

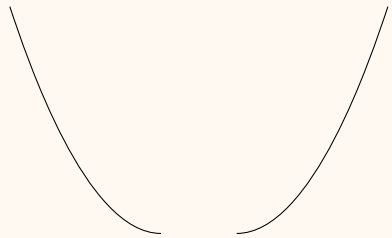


2.1.9.　放物線を描く

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。

基本

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

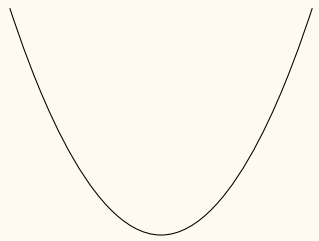


基本的には放物線の半分を描くイメージになります。\draw コマンド内で座標を 2 つ指定し、それらの間を parabola オペレーションでつなぎます。始めに指定した点を頂点とし、2 つ目に指定した点を終点とする放物線が描かれます。bend at end オプションを指定すると、始めに指定した点を始点とし、2 つ目に指定した点を頂点とする放物線が描かれます。

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

```
\begin{tikzpicture}
    \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
\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。

2.1.10.　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。

基本

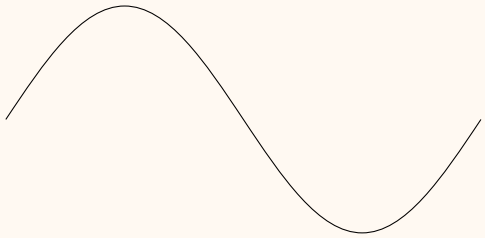
```
\begin{tikzpicture}
    \draw (0,0) sin (1,1);% (0,0) を位相 0 度、(1,1) を位相 90 度とする sin 関数
    \draw (2,1) cos (3,0);% (2,1) を位相 0 度、(3,0) を位相 90 度とする cos 関数
\end{tikzpicture}
```

基本的には $[0, \pi/2]$ の区間の sin 関数および cos 関数を描くイメージになります。\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 度、2 つ目に指定した点を位相 90 度とするサインカーブが描かれます。

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。

基本

```
\begin{tikzpicture}
    \draw (0,0) sin (pi/2,1.5) cos (pi,0) sin (3*pi/2,-1.5) cos (2*pi,0);% 1 周期の sin 関数
\end{tikzpicture}
```



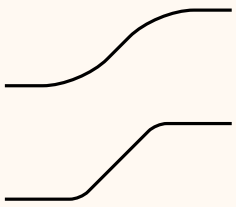
上の例のように、一つのパスでサインカーブを連続的につなぐ記述ができることに注意してください。また pi は TikZ の定義済み定数であり、円周率を表します。

2.1.11.　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に丸みを持たせたい場合は、rounded corners オプションを指定します。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。デフォルト値は 4pt です。

基本

```
\begin{tikzpicture}[very thick]
    \draw[rounded corners=5mm] (0,0) -- (1,0) -- (2,1) -- (3,1);% 半径 4mm
    \draw[rounded corners, yshift=-1.5cm] (0,0) -- (1,0) -- (2,1) -- (3,1);% 半径 4pt
\end{tikzpicture}
```

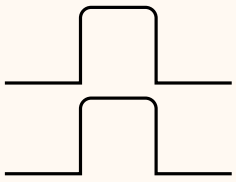


1 番目の例では (1,0) および (2,1) の接続箇所で丸められているのが分かります。

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

```
\begin{tikzpicture}[very thick]
    % (1,1) と (2,1) の接続部は丸くつなげる
    \draw (0,0) -- (1,0)[rounded corners] -- (1,1) -- (2,1)[sharp corners] --
    (2,0) -- (3,0);
    \draw[yshift=-1.2cm] (0,0) -- (1,0){[rounded corners] -- (1,1) -- (2,1)} --
    (2,0) -- (3,0);
\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。

長方形のような図形でも rounded corners オプション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

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

ほかに line join オプションを指定して、接合部の形状を指定することもできます。値は round, bevel, miter が選べます。デフォルト値は miter です。

接合部の形状を指定

```
\begin{tikzpicture}[line width=10pt]
    \draw[line join=round] (0,0) -- (0.5,1) -- (1,0);% round
    \draw[yshift=-1.5cm,line join=bevel] (0,0) -- (0.5,1) -- (1,0);% bevel
    \draw[yshift=-3cm,line join=miter] (0,0) -- (0.5,1) -- (1,0);% miter
\end{tikzpicture}
```

2.1.12.　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

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

背景色で引いた線の線幅の分だけ、交点において空白になるのが分かります。

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

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

preaction オプションを指定すると、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,draw=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。

2.2.　座標編

この節では、座標を指定する方法を解説します。

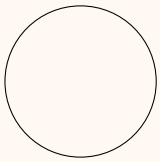
2.2.1.　座標の指定

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、鉛直方法を 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。

直交座標は `(x,y)` のように指定します。$x = 2\mathrm{cm}, y = 3\mathrm{cm}$ ならば `(2cm,3cm)` です。極座標は `(theta:r)` のように指定します。$r = 5\mathrm{cm}, \theta = 30°$ ならば `(30:5cm)` です。θ の取りうる範囲は $-360° \leq \theta \leq 720°$ です。なお角度をラジアンで指定したい場合は `(pi/6 r:5cm)` 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、計算式で指定することもできます。

基本

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



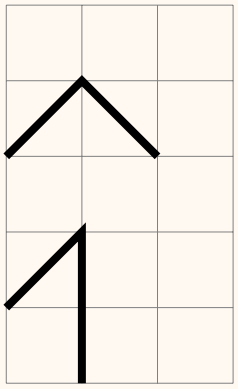
○

2.2.2.　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

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



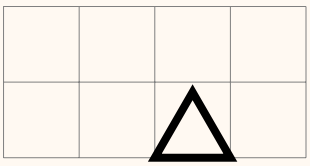
++ を使った場合は、移動後の座標が次のカレントポジションになります。上の (0,3) -- ++(1,1) -- ++(1,-1) では $(0,3)$ から $(1,1)$ だけ移動した $(1,4)$ までを線（--）で結びます。同時に $(1,4)$ が新たなカレントポジションになるため、そこから $(1,-1)$ だけ移動した $(2,3)$ までを線で結びます。

+ 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) -- +(1,1) -- ++(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 $(1,0)$ までを（$(1,2)$ 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

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

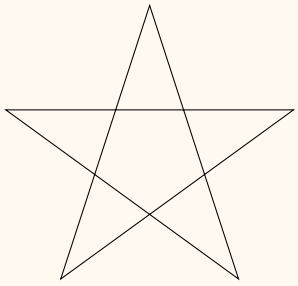


2.2.3.　関数を使った座標指定

座標指定するのに計算式や pi 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

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



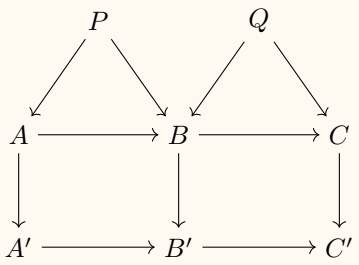
計算式を波カッコ { } で囲むよう心がけましょう。波カッコがなくてもうまく式評価してくれることもありますが、うまくコンパイルできないという時は波カッコで解決することもあります。

2.2.4.　マトリックス状に配置する

厳密には座標指定ではないですが、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

```
\begin{tikzpicture}
    \matrix [row sep=1cm, column sep=0.5cm] {
        & \node (P) {$P$}; & & \node (Q) {$Q$}; &\\
        \node (A) {$A$}; & & \node (B) {$B$}; & & \node (C) {$C$};\\
        \node (Ap) {$A'$}; & & \node (Bp) {$B'$}; & & \node (Cp) {$C'$};\\
    };
    \draw[->] (P) -- (A); \draw[->] (P) -- (B);
    \draw[->] (Q) -- (B); \draw[->] (Q) -- (C);
    \draw[->] (A) -- (Ap); \draw[->] (B) -- (Bp); \draw[->] (C) -- (Cp);
    \draw[->] (A) -- (B); \draw[->] (B) -- (C);
    \draw[->] (Ap) -- (Bp); \draw[->] (Bp) -- (Cp);
\end{tikzpicture}
```

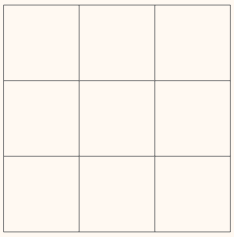


上の例では `\matrix` コマンドでノードをマトリックス状に配置しています。このマトリックスは 3 行 5 列になっています。中の要素の記述の仕方は LaTeX の `tabular` 環境と似ていますね。

2.2.5.　座標平面にグリッド線を引く

基本

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

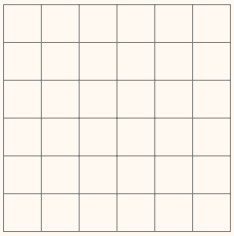


この例では (0,0) と (2,3) を対角とする長方形を描画し、中にグリッド線を引いています。help lines はスタイルの一種であり、line width=0.2pt, gray と等価です。したがって、自分好みの線幅や色でグリッド線を引くこともももちろん可能です。

方眼の間隔を指定するには step オプションを指定します。

方眼の間隔

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

2.3.　色編

この節では、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。
なお、紙媒体で読んでいる読者には、印刷の関係上着色されていないことご容赦ください。

2.3.1.　線の色を指定する

基本

```
\begin{tikzpicture}
    \draw[red] (0,0) -- (0,2) -- (1,1);% 赤で着色
\end{tikzpicture}
```

2.3.2.　内部を塗りつぶす

いわゆる内部を塗りつぶすという意味では次のようにします。

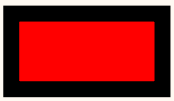
基本

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

しかし、通常は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します。

外周も描く

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



上の例では線の色が 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指定すると、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

```
\begin{tikzpicture}[line width=6pt]
    \draw[fill=red] (0,0) -- (0,1) -- (2,1) -- (2,0);% 閉じていないパス
\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パスとして指定した通り、(2,0) と (0,0) の間を結んでいないことが分かります。

閉じていないパスに対する塗りつぶし

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

交差するパスでも塗りつぶしができます。

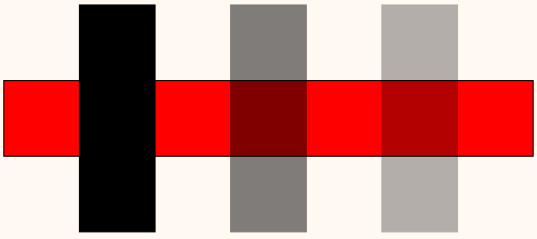
交差するパス

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

透明度を指定することができます。

透明度の指定

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



2.3.3.　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。

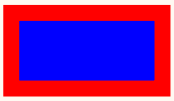
まず基本的な色を挙げましょう（紙媒体で読んでいる読者には、印刷の関係上着色されていないことご容赦ください。）。なお最新の情報は xcolor パッケージの説明を参照ください。

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			

描画のオプション引数として色名を指定します。

基本

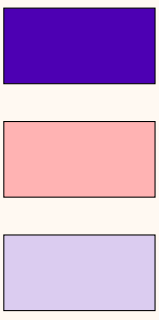
```
\begin{tikzpicture}[line width=6pt]
    \draw[red, fill=blue] (0,0) rectangle (2,1);% 線の色を赤、塗りつぶしを青
\end{tikzpicture}
```



またこれらの色を混ぜ合わせた色を作ることもできます。

色の混合

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

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}
```

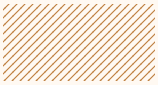
tikzpicture 環境外で記述します。xcolor パッケージは TikZ とは独立ですから、この色名は tikzpicture 環境外でも使用できます。

2.3.4.　内部をパターンで塗りつぶす

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

基本

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



pattern オプションにパターンを、pattern color オプションに色を指定します。
パターンとして以下があります。

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	

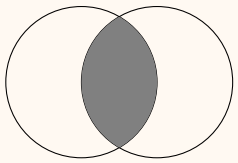
2.3.5.　図形の一部だけ塗りつぶす

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場合、以下のようにします。

集合の共通部分

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



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 環境で囲んでいます。

2.3.6.　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。

ノードの色

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

色のみを指定（color= を省略したことに相当）すると輪郭とテキストにその色が反映されます。text= オプションでテキストの色を指定できます。fill オプションでノードの内部を塗りつぶすときの色を指定できます。

なおパスの途中で複数のノードを配置した場合、後続のノードの色指定を省略すると、それより前のノードへの色指定が後続のノードに継承されます。

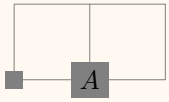
2.4.　ノード・ラベル編

この節では、ノードやラベルを作成する方法を解説します。

2.4.1.　ノードを配置する

基本

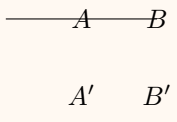
```
\begin{tikzpicture}
    \draw[help lines] (0,0) grid (2,1);
    \node[fill=gray] at (0,0) {};% ノードのみを配置
    \node[fill=gray] at (1,0) {$A$};% ラベルを持つノード
\end{tikzpicture}
```



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、次に述べる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。

複数のラベル

```
\begin{tikzpicture}
    \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
    \draw (0,-1) (1,-1) node {$A'$} (2,-1) node {$B'$};% ラベルのみを配置
\end{tikzpicture}
```



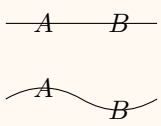
座標指定の後に node オペレーションを記述すると、その座標の位置にノードを配置します。座標を明示的に指定したい場合は at を使います。

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、実際は同じものです。

線分や曲線の途中にラベルを配置することもできます。

線分の途中にラベルを配置

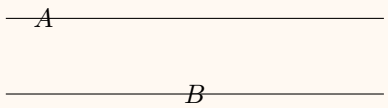
```
\begin{tikzpicture}
    \draw (0,0) -- node {$A$} (1,0) -- node {$B$} (2,0);% 線分の途中にラベルを配置
    \draw (0,-1) to[bend left=30] node {$A$} (1,-1) to[bend right=30] node {$B$}
    (2,-1);% 曲線の途中にラベルを配
    置
\end{tikzpicture}
```



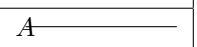
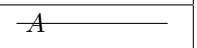
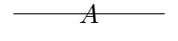
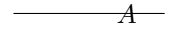
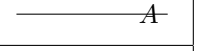
座標指定の前に node オペレーションを記述すると、前の座標と今の座標の中間位置にノードを配置します。少し紛らわしいですが、座標指定の前と後とで node の挙動が異なることに注意してください。中間位置ではなく、任意の相対位置にノードを配置するには pos オプションを使います。引数には 0 から 1 の間の数を与えます。

相対位置を指定

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



上の例のように pos=0.1 と指定すると、(0,0) から (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 は pos=0.5 と同等です。他に以下のものがあります。

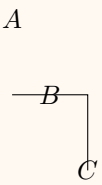
オプション	意味	例
at start	pos=0	
very near start	pos=0.125	
near start	pos=0.25	
midway	pos=0.5	
near end	pos=0.75	
very near end	pos=0.875	
at end	pos=1	

2.4.2.　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。

基本

```
\begin{tikzpicture}
    \node (A) at (0,0) {$A$};% A と命名
    \draw (0,-1) -- node (B) {$B$} (1,-1) -- (1,-2) node (C) {$C$} ;% パス途中での命名
    \coordinate (D) at (1,-3);% 大きさ 0 のノード
\end{tikzpicture}
```



node の後に（**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。

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。上の例の 3 つ目がそれにあたります（実行結果上は見えませんが）。node では命名を省略することもできますが、\coordinate で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

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

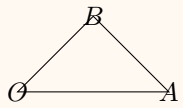
座標を指定する代わりに（**ノード名**）と指定します。

2.4.3.　オプションによるノードの位置指定

線分を引きつつラベルを配置すると、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

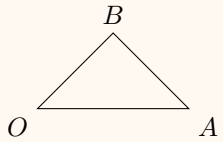
```
\begin{tikzpicture}
    \draw (0,0) node {$O$} -- (2,0) node {$A$}  -- (1,1) node {$B$} -- cycle;% 座標と同じ位置にノードを配置
\end{tikzpicture}
```



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` のオプションを指定します。

隣接する位置

```
\begin{tikzpicture}
    \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
\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` はノードから見たラベルの表示位置ではなく、$(0,0)$ からのノードそのものの配置位置を指定するオプションです。

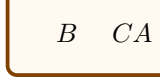
配置に関するオプションには以下があります。

オプション	位置
above	上
below	下
left	左
right	右
above left	左上
above right	右上
below left	左下
below right	右下
centered	中央

node[above of=**ノード名**] と指定すると隣接させたいノードを明示的に指定できます。

隣接するノードを指定

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

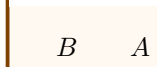


上の例ではいずれも A の左にノードを配置しています。1 つ目は left of、2 つ目は at で指定しています。見て分かるように隣接の距離に違いがあります。

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。

隣接するときの距離

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



隣接するノードとその距離の同時指定は positioning ライブラリを読み込めば可能です。

隣接するノードと距離を同時に指定

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

一方、同じ効果をもたらすオプションとして anchor があります。

アンカーによる指定

```
\begin{tikzpicture}
    \node (A) at (0,0) {$A$};% 起点となるノード
    \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
    。
\end{tikzpicture}
```

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「新しいノードの下位置（south）を A 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。？？？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

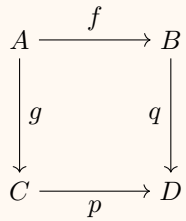
```
\begin{tikzpicture}
    \node (A) at (0,0) {$A$};% 起点となるノード
    \node[anchor=south west] at (A) {$B$};% 新しいノードの左下のアンカーを A の位置に合わせる。
    \node[anchor=north east] at (A) {$C$};% 新しいノードの右上のアンカーを A の位置に合わせる。
\end{tikzpicture}
```

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。

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 オプションを指定します。

パスの途中のノード

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



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、右側に配置するには p と q のように swap オプションを指定します。

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 オプションを使います。

曲線に沿うラベル

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

2.4.4.　ノードの形状、大きさ

node のオプションとして draw を指定するとノードの輪郭が描かれます。このことからノードは形状と大きさを持つことが分かります。

ノードの輪郭を描く

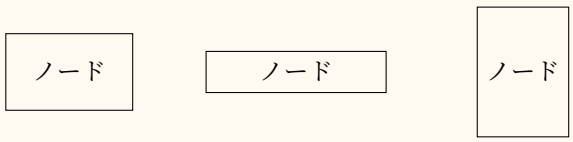
```
\begin{tikzpicture}
    \node[draw, rectangle] at (0,0) {ノード};% 長方形
    \node[draw, circle] at (3,0) {ノード};% 円
    \node[draw] at (6,0) {ノード};% デフォルトでは rectangle
\end{tikzpicture}
```

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。実は coordinate というのもありますが、これは \coordinate と同じ結果になるにすぎません。coordinate は大きさも形状もないノードです。

輪郭とテキストの間隔を制御することができます。

輪郭とテキストの間隔

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

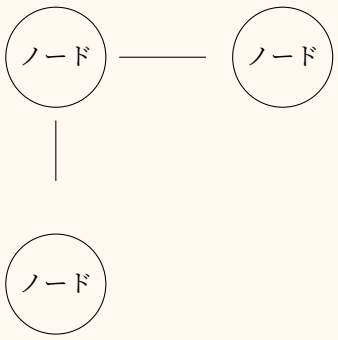


inner sep は上下左右、inner xsep は左右、そして inner ysep は上下の間隔を制御します。CSS で言うところの padding と同じ概念にあたります。

一方、輪郭とその外との間隔を制御することもできます。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

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

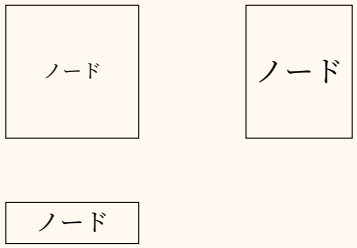


このオプションの使いどころは、主にアンカーの位置を制御したいときです。上の例では 3 つのノードの間を線で結んでいますが、線の端点がノードに接していないことが見て取れます。これは線の接続先であるアンカーが外側に移動したためです。outer sep は上下左右、outer xsep は左右、そして outer ysep は上下の間隔を制御します。

ノードの形状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、輪郭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輪郭の大きさを確保することで解決できます。

輪郭の最小値

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

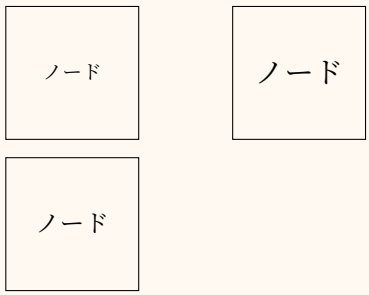


minimum size は幅と高さ、minimum height は高さ、そして minimum width は幅の最小値を指定します。

輪郭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。

輪郭の最小値（スタイル設定）

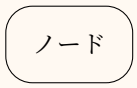
```
\begin{tikzpicture}[box/.style={draw, minimum size=50pt}]% 幅と高さの最小値をスタイル設定
    \node[draw, box] (A) at (0,0) {\footnotesize{ノード}};
    \node[draw, box] (B) at (3,0) {\large{ノード}};
    \node[draw, box] (C) at (0,-2) {ノード};
\end{tikzpicture}
```



rectangle の場合のオプションとして rounded corners を追加すると角が丸くなります。

角が丸い長方形

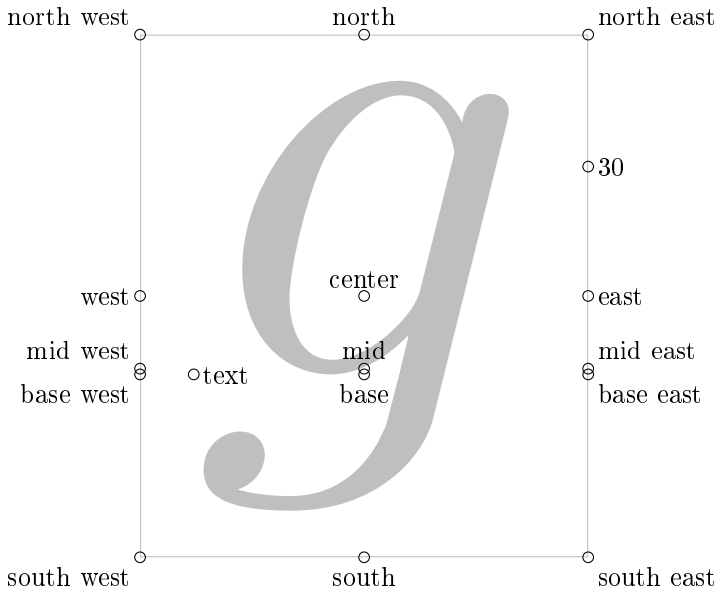
```
\begin{tikzpicture}
    \node[draw, rounded corners=10pt, inner sep=10pt] (A) at (0,0) {ノード};% 角が丸い長方形
\end{tikzpicture}
```



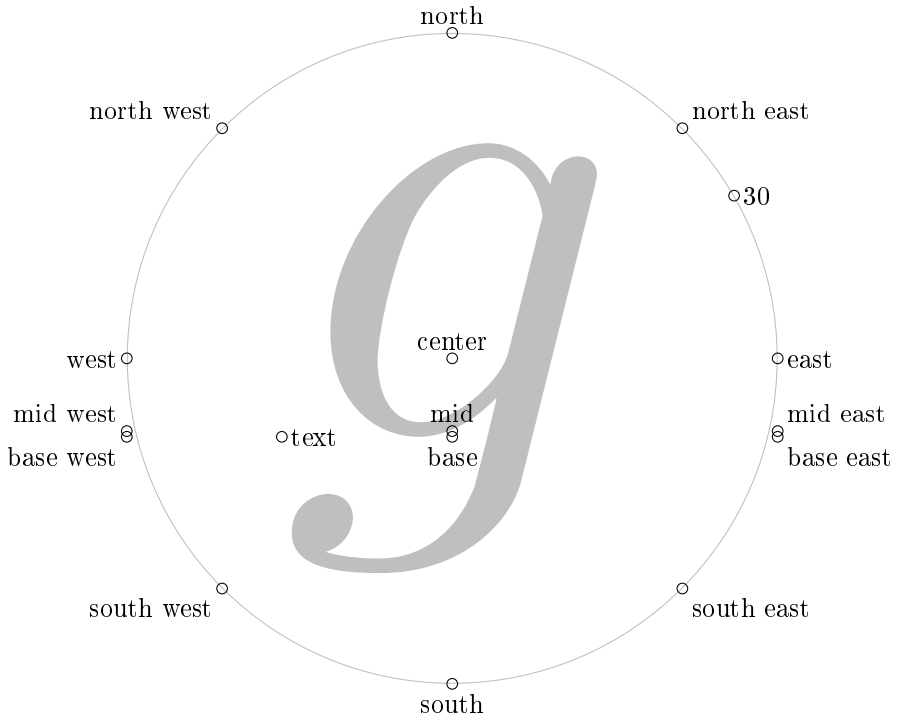
2.4.5.　アンカーの種類

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



A という名前のノードに対して `A.center` と記述すると、ノードの形状に応じて前掲の `center`

の位置を指し示すことになります。また A.30 は、A.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。

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

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

O.baseline の位置を中心に円を描きたい場合は

ノード O のベースの位置に円を描く

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

とします。

一方、新しいノードを作成するときに anchor オプション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

```
\begin{tikzpicture}
    \draw (0,0) circle[radius=2pt];% 目印をつけた
    \node[draw] (O) at (0,0) {\Huge O};% (0,0) の位置にノード O を配置する
\end{tikzpicture}
```

(0,0) の位置と O.base の位置が一致するようにノードを配置したい場合は

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

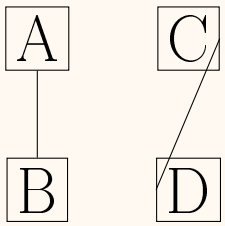
```
\begin{tikzpicture}
    \draw (0,0) circle[radius=2pt];% 目印をつけた
    \node[draw, anchor=base] (O) at (0,0) {\Huge O};% (0,0) の位置にノード O を配置する
\end{tikzpicture}
```

とします。

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

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



2.4.6.　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ので、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。一方、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。

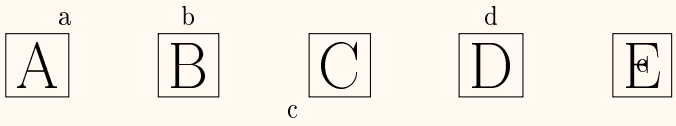
基本

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

この例ではラベル A の右にラベル B を label オプションを使って配置しています。label=**角度:テキスト** の形式で記述します。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

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

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

```
\begin{tikzpicture}
    \node[draw] at (0,0) [label={[red, draw, circle] above:{\Huge a}}]{A};% 円形の
    赤いラベル
\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。

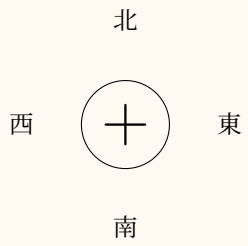
ラベルの複数配置

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

label distance オプション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

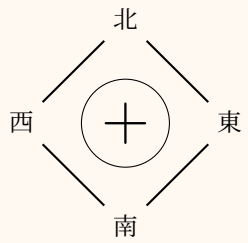
```
\begin{tikzpicture}[label distance=5mm]
    \node[draw, circle] at (0,0)
        [label=north:北, label=west:西, label=south:南, label=east:東]
        {\Huge $+$}; % ラベル東西南北を距離を取って配置
\end{tikzpicture}
```



label オプションで配置したラベルもやはりノードの一種ですので、ノード名をつけることができます。

ノードに命名

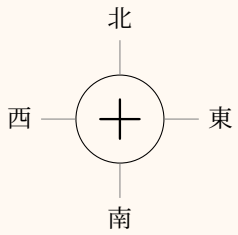
```
\begin{tikzpicture}[label distance=5mm]
    \node[draw, circle] at (0,0)
        [label={[name=kita] north:北},
         label={[name=nishi] west:西},
         label={[name=minami] south:南},
         label={[name=higashi] east:東}]
         {\Huge $+$}; % ラベルのノードに命名

    \draw[thick] (kita) -- (nishi) -- (minami) -- (higashi) -- (kita);
\end{tikzpicture}
```



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

ラベルへ線を引く

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



quotes ライブラリを読み込めばラベルの配置がもっと簡潔にできます。

ラベルを簡潔に配置

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

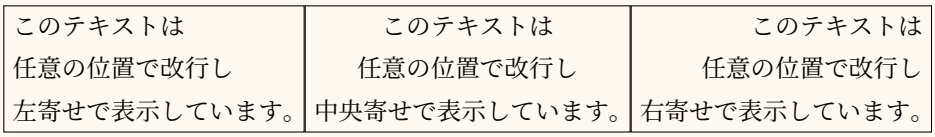
label オプションの代わりに、**"ラベル" オプション** と記述します。ラベルについてのオプションが複数ある場合は波カッコで囲みます。

2.4.7.　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方法を解説します。

基本

```
\begin{tikzpicture}
    \node[draw, align=left] (A)
    {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
    \node[draw, align=center, anchor=south west] (B) at (A.south east) {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
    \node[draw, align=right, anchor=south west] (C) at (B.south east) {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
\end{tikzpicture}
```

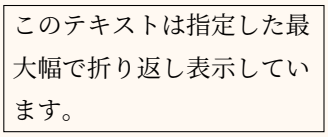


改行は LaTeX での文法と同様 \\ で行います。align オプションでテキストの位置を揃え、left が左寄せ、center が中央揃えそして right が右揃えになります。align オプションを指定しないと改行がされないことに注意しましょう。

一方、テキストの幅を指定してあげ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

```
\begin{tikzpicture}
    \node[draw, text width=4cm]
    {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。};% 左寄せ
\end{tikzpicture}
```

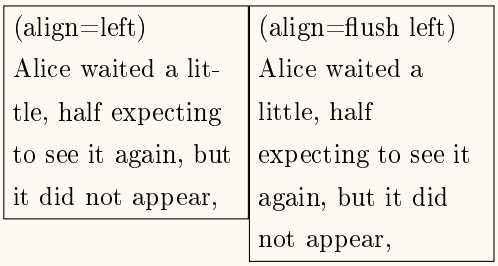


テキストの幅を text width オプションで指定します。このときは align オプションは必須ではなく、既定値は left になります。

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、それに応じて align オプションに left または flush left を指定します。

改行の仕方の指定

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



上の例[*1] の左が left、右が flush left で位置揃えしたものです。

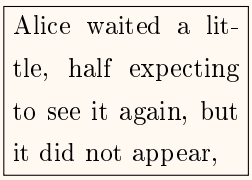
left はハイフネーションによる単語分割をしてでも右端のバランスをとろうとします。flush left は単語分割をしません。

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

align=justify は左右両端で位置揃えします。

左右両端で位置揃え

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

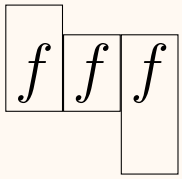


テキストの高さを指定するには text height、text depth を使います。

[*1] Lewis Carroll, ALICE'S ADVENTURES IN WONDERLAND

左右両端で位置揃え

```
\begin{tikzpicture}
    \node[draw] (A) {\Huge $f$};
    \node[draw, anchor=base east, text height=1cm] at (A.base west) {\Huge $f$};
    \node[draw, anchor=base west, text depth=1cm] at (A.base east) {\Huge $f$};
\end{tikzpicture}
```



上の例の真ん中が高さを指定ししなかった場合、左が text height、右が text depth による指定の結果です。

2.4.8.　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。

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

```
\begin{tikzpicture}
    \coordinate (A) at (0,0);\draw[fill] (A) circle[radius=1pt];
    \coordinate (B) at (2,0);\draw[fill] (B) circle[radius=1pt];

    \draw (A) ++(0,-0.1) -- ++(0,-0.1) coordinate (A1) -- ++(0,-0.1);
    \draw (B) ++(0,-0.1) -- ++(0,-0.1) coordinate (B1) -- ++(0,-0.1);
    \draw[<->] (A1) -- node[inner sep=2mm, fill=white] {2cm} (B1);
\end{tikzpicture}
```

2.5.　平行移動、回転、拡大縮小編

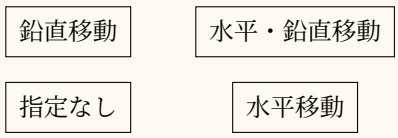
この節では、図形全体あるいは図形の一部を、平行移動、回転および拡大縮小する方法を解説します。

2.5.1.　平行移動

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

基本

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



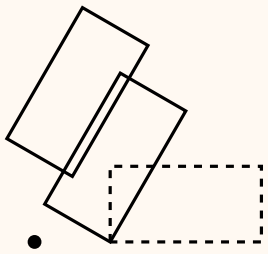
xshift オプションと yshift オプションではそれぞれの移動量を指定します。shift オプションでは shift={(**水平方向の移動量**, **鉛直方向の移動量**)} と指定します。

2.5.2.　回転

原点を中心とする回転は rotate オプション、回転中心も指定したい場合は rotate around を指定します。

基本

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



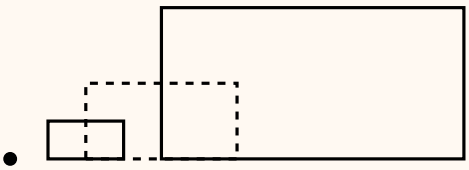
rotate オプションでは回転角を度で指定します。rotate around では rotate around={**角度**:(**回転の中心点**)} と指定します。

2.5.3.　拡大縮小

拡大縮小は scale オプションまたは scale around オプションを指定します。scale オプションは原点を中心とした拡大縮小を行います。引数に倍率を指定します。

基本

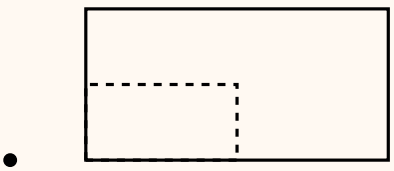
```
\begin{tikzpicture}[very thick]
    \draw[fill](0,0) circle[radius=2pt];
    \draw[dashed] (1,0) rectangle (3,1);% 拡大縮小前
    \draw[scale=2] (1,0) rectangle (3,1);% 原点を中心として 2 倍に拡大
    \draw[scale=0.5] (1,0) rectangle (3,1);% 原点を中心として半分に縮小
\end{tikzpicture}
```



拡大縮小の中心を指定する場合は scale around オプションを指定します。scale around={倍率:(拡大の中心点)} と指定します。

中心を指定した拡大縮小

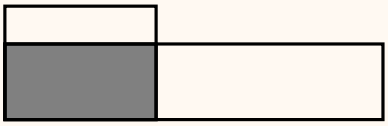
```
\begin{tikzpicture}[very thick]
    \draw[fill](0,0) circle[radius=2pt];
    \draw[dashed] (1,0) rectangle (3,1);% 拡大縮小前
    \draw[scale around={2:(1,0)}] (1,0) rectangle (3,1);% (1,0) を中心として 2 倍に拡
    大
\end{tikzpicture}
```



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 オプションおよび yscale オプションを指定します。

水平方向のみおよび鉛直方向のみの拡大縮小

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



通常 scale オプションは tikzpicture 環境のオプションとして指定する場合が多いようです。

2.6.　作図風コマンド編

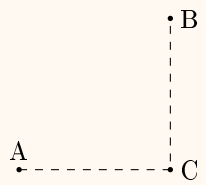
この節では、幾何学的な考え方で作図を行う方法を解説します。

2.6.1.　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 から水平線を、B から鉛直線を引いたときの交点は (A-|B) で、A から鉛直線を、B から水平線を引いたときの交点は (A|-B) で取得できます。取得結果はいずれも座標であり、コマンド内であたかもノード名で座標指定するように使用することができます。

基本

```
\begin{tikzpicture}
    \coordinate (A) at (0,0);
    \coordinate (B) at (2,2);
    \fill (A) circle[radius=1pt] node[above] {A};
    \fill (B) circle[radius=1pt] node[right] {B};

    \fill (A -| B) circle[radius=1pt] node[right] {C};% 交点に黒点を描画
    \draw[dashed] (A) -- (A -| B) -- (B);% 交点を経由する直線
\end{tikzpicture}
```



(A|-B) の中でノード名を指定する代わりに座標を直接指定することもできます。

基本

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

2.6.2.　任意のパスの交点

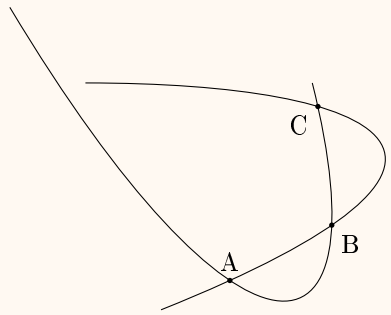
intersections ライブラリを使うことで、任意の曲線の交点を求めることができます。
交点を扱うには次の手順を踏むことになります。

1. 2 つのパスを定義し、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

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



1 行目と 2 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 オプションでパスに名前をつけています。3 行目では name intersections オプションで交点を求め、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 オプションの引数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。ノード名の定義を省略すると、Ti*k*Z が自動的に intersection-1、intersection-2 というノード名をつけます。

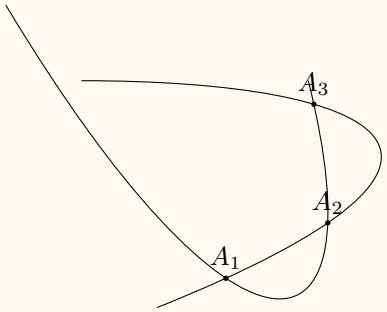
by オプションで指定できるのはノード名のみであり、それぞれのノード名をどの交点につけるかまでは指定することはできません。

交点の数があらかじめ分かっている場合は by オプションでノード名を定義できますが、分からない場合は Ti*k*Z 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です。あとは \foreach を併用して、ループを回すことになるでしょう。

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

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

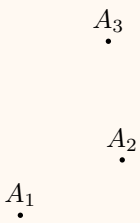


この例ではマクロ \total に交点数が格納されています。また交点のノード名は is-1、is-2、is-3 となります。

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 の代わりに \path を使います。

曲線を描画しない場合

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

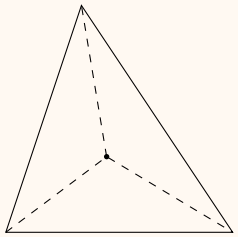


2.6.3.　位置ベクトルの和とスカラー倍を使った座標指定

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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \draw (0,0) coordinate (A) -- (3,0) coordinate (B) -- (1,3) coordinate (C)
    -- cycle;

    \fill ($1/3*(A) + 1/3*(B) + 1/3*(C)$) circle[radius=1pt] coordinate (D);% 重
    心
    \draw[dashed] (A) -- (D) -- (B) (D) -- (C);
\end{tikzpicture}
```

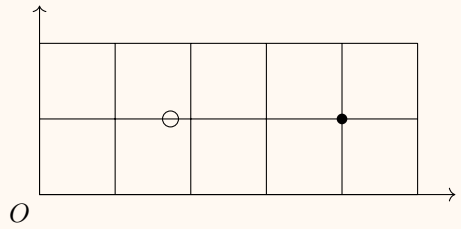


上の例では三角形の重心に黒点を描画しています。重心は $(\vec{OA}+\vec{OB}+\vec{OC})/3$ で求められますので、それに対応して ($1/3*(A) + 1/3*(B) + 1/3*(C)$) で重心の座標を表現しています。

($...$) 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。($2*cos(30)*(0,1) + 2*in(30)*(1,0)$) 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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \draw (0,0) grid (5,2);
    \draw[<->] (0,2.5) -- (0,0) node[below left] {$O$} -- (5.5,0);
    \fill ($2*(2,1) - (0,1)$) circle[radius=2pt];
    \draw ($2*cos(30)*(1,0) + 2*sin(30)*(0,1)$) circle[radius=3pt];
\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。`($1/3*(A) + 1/3*(B) + 1/3*(C)$)` は正しい文ですが、`($1/3*((A) + (B) + (C))$)` はエラーになります。数学で言うところの分配則は使えません。

2.6.4.　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。

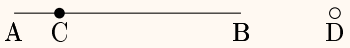
AB 間を $t : 1-t$ で内分・外分する場合は、($(A の座標)!t!(B の座標)$) の形式で記述します。($...$) で囲むことに注意しましょう。$0 < t < 1$ の場合は内分、$t < 0$ または $1 < t$ の場合は外分になります。

xcolor での比率の指定の仕方とは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。また内分の場合 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

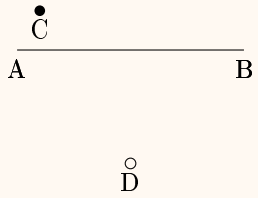
```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,0);
    \draw (A) node[below] {A} -- (B) node[below] {B};

    \fill ($(A)!.2!(B)$) circle[radius=2pt] node[below] {C};% 0.2:0.8 の内分点
    \draw ($(0,0)!sqrt(2)!(3,0)$) circle[radius=2pt] node[below] {D};% 1:√2 の外分点
\end{tikzpicture}
```



($(A の座標)!t!角度:(B の座標)$) 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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,0);
    \draw (A) node[below] {A} -- (B) node[below] {B};

    \fill ($(A)!.2!60:(B)$) circle[radius=2pt] node[below] {C};% 60 度回転して 0.2:0.8 で内分
    \draw ($(0,0)!1/sqrt(2)!atan(-1):(3,0)$) circle[radius=2pt] node[below] {D};% -45 度回転して 1:√2 で外分
\end{tikzpicture}
```

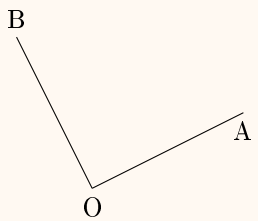


上の例のラベル C のノードの場合は、まず A を中心として B を 60 度だけ仮想的に回転します。そして A と（仮想的な）B との間を 0.2 : 0.8 で内分し、そこに黒点を描画しています。

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あるでしょう。

与えられた直線への垂直線

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



2.6.5.　指定した 2 点間の指定した距離の座標

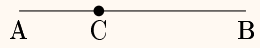
与えられた 2 点 A と B の間の、A からの指定距離の座標を取得することができます。このとき calc ライブラリを使います。

($(A の座標)!距離!(B の座標)$) 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。($...$) 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つけます。単位を付け忘れると 2.6.4 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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,0);
    \draw (A) node[below] {A} -- (B) node[below] {B};

    \fill ($(A)!30pt!(B)$) circle[radius=2pt] node[below] {C};% A から 30pt の距離
\end{tikzpicture}
```



($(A の座標)!距離!角度:(B の座標)$) 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。

回転してから距離を測る

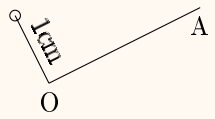
```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,0);
    \draw (A) node[below] {A} -- (B) node[below] {B};

    \fill ($(A)!30pt!60:(B)$) circle[radius=2pt] node[below] {C};% 60 度回転してから A から 30pt の距離
\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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:O] (O) at (0,0);
    \coordinate[label=below:A] (A) at (2,1);
    \draw (O) -- (A);
    \draw (O) -- node[right, sloped, above] {1cm} ($(O)!1cm!90:(A)$)
    circle[radius=2pt, fill];% 垂直
    線
\end{tikzpicture}
```



2.6.6.　垂線との交点

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。

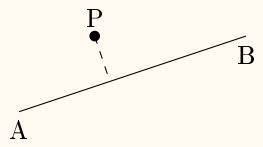
このとき calc ライブラリを使います。

($(A の座標)!(P の座標)!(B の座標)$) 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 への垂線との交点の座標を取得できます。($...$) 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

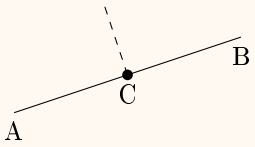
```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,1);
    \coordinate[label=above:P] (P) at (1,1);
    \draw (A) -- (B);
    \fill (P) circle[radius=2pt];

    \draw[dashed] (P) -- ($(A)!(P)!(B)$);% P から AB への垂線
\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 2.6.5 節の方法を使います。

与えられた点を通る垂線

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,1);
    \coordinate[label=below:C] (C) at ($(A)!0.5!(B)$);
    \draw (A) -- (B);
    \fill (C) circle[radius=2pt];

    \draw[dashed] (C) -- ($(C)!1cm!90:(B)$);% C からの垂線
\end{tikzpicture}
```



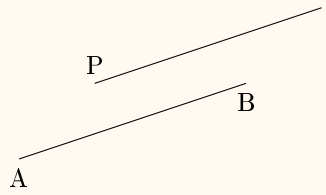
2.6.7.　平行線を引く

点 P と直線 AB が与えられたとき、P を通り AB と平行な直線を引くことができます。これは 2.6.3 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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,1);
    \coordinate[label=above:P] (d) at (1,1);
    \draw (A) -- (B);

    \draw ($(A)+(d)$) -- ($(B)+(d)$);
\end{tikzpicture}
```

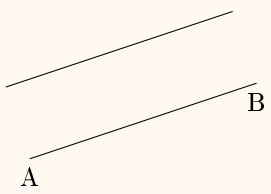


上の例では、ベクトル d を平行移動量として、ベクトル和を使って描画しています。

平行移動量が分からない場合は、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量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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \coordinate[label=below:A] (A) at (0,0);
    \coordinate[label=below:B] (B) at (3,1);
    \draw (A) -- (B);

    % 平行移動量を算出
    \coordinate (d) at ($(A)!1cm!90:(B) - (A)$);% AB から 1cm 隔てる

    \draw ($(A)+(d)$) -- ($(B)+(d)$);
\end{tikzpicture}
```



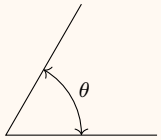
上の例の平行移動量を算出している部分に注目しましょう。まず A を中心として B を 90 度だけ仮想的に回転させ、A から仮想的な B に向かって 1cm だけ移動します。その点から A を引くと、求める移動量ベクトルとなります。これらをベクトルに対する演算として表現しています。

2.6.8.　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。

基本

```
%\usetikzlibrary{angles, quotes}
\begin{tikzpicture}
    \draw (2,0) coordinate (A) -- (0,0) coordinate (B) -- (60:2) coordinate (C);

    \draw pic["$\theta$", draw, <->, font=\footnotesize, angle eccentricity=1.2,
    angle radius=1cm] {angle=A--B--C};% 角
    度
\end{tikzpicture}
```

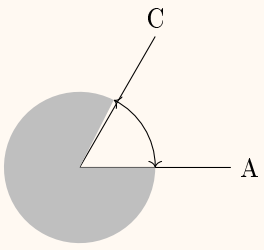


\draw pic {angle=A--B--C} の書式が基本です。このとき B の角度を表す記号が描画されます。A、B および C はノード名である必要があり、座標の直接指定はできません。

A--B--C の順序で A、B、C を記述した場合、直線 BA を出発し、反時計周りで回り、直線 BC に到達するように、角度の円弧が描画されます。

円弧の描画のしかた

```
%\usetikzlibrary{angles}
\begin{tikzpicture}
    \draw (2,0) node[right] (A) {A} -- (0,0) coordinate (B) -- (60:2)
    node[above] (C) {C};

    \draw pic[draw, <->, angle radius=1cm] {angle=A--B--C};% 角度
    \draw pic[fill=lightgray, angle radius=1cm] {angle=C--B--A};% 角度
\end{tikzpicture}
```



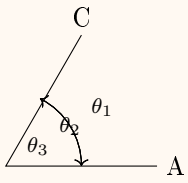
上の例では angle=A--B--C のほうが内角、angle=C--B--A のほうが外角を描画しています。

オプションには次のものがあります。draw 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 を指定しないと円弧が描画されません。塗りつぶしの fill や、矢印の <-> も使えます。angle radius オプション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 オプションで指定します。ラベルの配置位置は angle eccentricity オプションで指定します。以下に angle eccentricity オプションの例を示します。

ラベルの配置位置

```
%\usetikzlibrary{angles, quotes}
\begin{tikzpicture}
    \draw (2,0) node[right] (A) {A} -- (0,0) coordinate (B) -- (60:2)
    node[above] (C) {C};

    \draw pic["$\theta_1$", draw, <->, font=\footnotesize, angle
    eccentricity=1.5, angle radius=1cm] {angle=A--B--C};% 1.5
    \draw pic["$\theta_2$", draw, <->, font=\footnotesize, angle eccentricity=1,
    angle radius=1cm] {angle=A--B--C};% 1
    \draw pic["$\theta_3$", draw, <->, font=\footnotesize, angle
    eccentricity=0.5, angle radius=1cm] {angle=A--B--C};% 0.5
\end{tikzpicture}
```

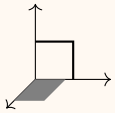


angle eccentricity の値が大きいと円弧の外側、小さいと内側に配置されることが分かります。1 で円弧と重なります。

直角記号もあります。

直角記号

```
%\usetikzlibrary{angles}
\begin{tikzpicture}
    \coordinate (O) at (0,0,0);
    \draw[<-] (1,0,0) coordinate (A) -- (O);
    \draw[<-] (0,0,1) coordinate (B) -- (O);
    \draw[<-] (0,1,0) coordinate (C) -- (O);

    \draw pic [fill=gray,angle radius=4mm] {right angle = A--O--B};% AOB 間の直角
    \draw pic [draw,thick] {right angle = A--O--C};% AOC 間の直角
\end{tikzpicture}
```

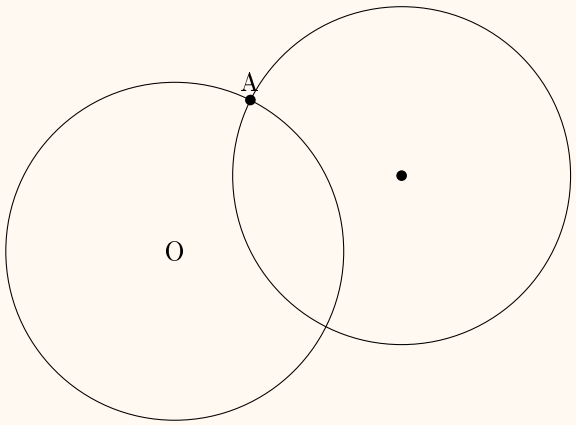


2.6.9.　与えられた中心を持ち、与えられた点を通る円

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。

基本

```
%\usetikzlibrary{through}
\begin{tikzpicture}
    \fill (1,2) circle[radius=2pt] coordinate[label=above:A] (A);

    \node[draw, circle through=(A)] at (0,0) {O};% 中心 (0,0) の A を通る円
    \fill (3,1) circle[radius=2pt] node[draw, circle through={(1,2)}] {}; % 中心 (3,1) の (1,2) を通る円
\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 オプションに経由したい座標を指定します。ノード名でも構いません。

2.6.10.　座標計算を連続して記述する

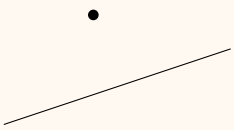
calc ライブラリを使った座標計算を連続して記述することができます。

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。

答えは ($(A)!.5!(B)!1cm!90:(B)$) 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

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \draw (0,0) coordinate (A) -- (3,1) coordinate (B);

    \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
\end{tikzpicture}
```

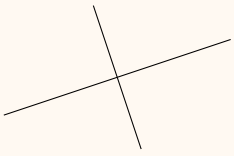


3 つ以上連続した座標計算ももちろんできます。

冒頭で述べた垂直二等分線を引いてみましょう。

垂直二等分線 1

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \draw (0,0) coordinate (A) -- (3,1) coordinate (B);

    \draw ($(A)!.5!(B)!1cm!90:(B)$)
          -- ($(A)!.5!(B)!-1cm!90:(B)$); % 垂直二等分線
\end{tikzpicture}
```



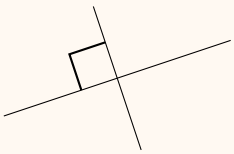
さて、これを見ると、座標計算の連続指定には限界があるなと思う方もいるかもしれません。一文で書けてすっきりしている反面、可読性には若干の難があります。またこの例のように、垂直二等分

線の 2 つの端点を求めるのに、似たような座標指定を 2 つ書いています。つまり一文で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）。

座標計算の連続指定と、適宜ノード名をつけながらの座標指定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

```
%\usetikzlibrary{calc, angles}
\begin{tikzpicture}
    \draw (0,0) coordinate (A) -- (3,1) coordinate (B);

    \draw ($(A)!.5!(B)$) coordinate (C) % 中点
          ($(C)!1cm!90:(B)$) coordinate (D) % 一方の端点
       -- ($(D)!2!(C)$) % 垂直二等分線
           pic [draw, thick] {right angle = A--C--D};
\end{tikzpicture}
```



2.6.11.　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 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。

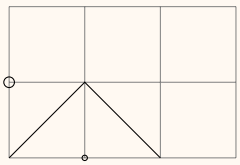
基本的な構文は

\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           (0,0) -- (\p1) -- (\p2) % レジスタの値を使って線分を引く
7           (\x1,0) circle[radius=\n1 pt] % \p1 の x 成分に半径\n1 の円を描く
8           (0,\y1) circle[radius={\n2 + 1pt}];% \p1 の y 成分に半径\n1 + 1pt の円を描く
9 \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 4 行目で線分を引いています。

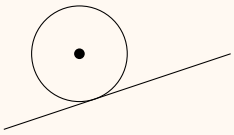
\n で始まるレジスタには数値が格納されます。上の例では円の半径として使われています。\n1 のようにただの数値を格納することもできますし、\n2 のように単位つきの数値を格納することもできます。

\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 \p1 の x 成分と y 成分を表す円の位置の指定に使われています。

例として、直線 AB と点 O が任意に与えられたときの、O を中心とし AB と接する円を作図してみましょう。

直線と接する円

```
%\usetikzlibrary{calc}
\begin{tikzpicture}
    \draw (0,0) coordinate (A) -- (3,1) coordinate (B);
    \fill (1,1) circle[radius=2pt]  coordinate (O);

    \draw let \p1 = ($(A)!(O)!(B) - (O)$), % 半径を表すベクトル
              \n1 = {veclen(\x1,\y1)} % 半径
          in (O) circle[radius=\n1]; % 求めた半径の円
\end{tikzpicture}
```



veclen(a, b) は数学関数であり $\sqrt{a^2+b^2}$ を返します。

(A)!(O)!(B) は O から AB に下ろした垂線と AB との交点の座標ですので、(A)!(O)!(B) - (O) はその交点と O 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。

レジスタ名は \p1、\n1 など、\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。

2.7.　スタイル編

この節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ます。TikZ ではスタイル自身が一つのキーとして扱われます。オプション指定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（キー）を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。複数のオプションがいくつも並んでいるとコードが見づらいものです。同じオプションをいくつも指定するのならば、スタイル名を一つ記述するだけのほう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くなった場合は、デザイン定義の部分だけを修正すれば済みます。もし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。

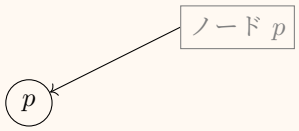
たとえ同じオプション指定であっても、使われるところが違うのであれば互いに異なるスタイル名とすべきです。写像も「ならば」も同じ単線の矢印で表現するという場合であっても、スタイル名は `mapping`、`then` などと別名をつけておきます。あとから「ならば」だけ二重線にしたくなったら、`then` スタイルだけ変更すれば済みます。

2.7.1.　スタイル定義

スタイル定義は以下のように行うのが基本です。

基本

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



tikzpicture 環境のオプション指定を行う箇所で **スタイル名**/.style={**オプション指定**} と記述します。この例では gray, draw の二つのオプション指定に annot というスタイル名をつけています。注釈（annotation）を表示する \node 内で使うことを想定して定義しました。

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を描画しています。

tikzpicture 環境でスタイルを定義すると、その定義は今の tikzpicture 環境内だけで有効です。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

```
% プリアンブル部
\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。

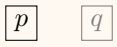
ローカルなスタイル

```
\begin{scope}[annot/.style={gray, draw}]

\end{scope}
```

スタイルを使用する際、別のオプション指定を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

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



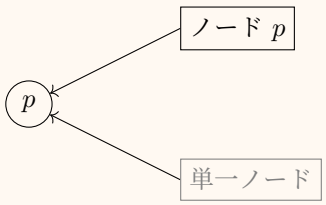
この例を見ると、\node[annot, black] では black をあとに記述しているので、黒が有効です。\node[black, annot] では black を先に記述しているので、annot つまりグレーが有効です。一方、draw は活きているので輪郭は描画されます。

2.7.2.　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。

基本

```
\begin{tikzpicture}[annot/.style={color=#1, draw},
                    annot/.default=gray] %注釈のスタイル
    \node[draw, circle] (p) at (0,0) {$p$};
    \draw[<-] (p) -- (2,1) node[annot=black, right] {ノード $p$};
    \draw[<-] (p) -- (2,-1) node[annot, right] {単一ノード};
\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が、#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、かつその引数が #1 のところに代入されます。例では色を引数として取っていることになります。TEX 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。

2.7.3.　every 型のスタイル

ノードや線などの図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。例えば図中のノード全般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

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

この例ではすべてのノードに circle, draw をオプション指定しています。このオプションはすべてのノードに適用されるため、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

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

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。\node (p) 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。\node[except] (q) 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	意味
every picture	すべての図要素。\tikzset コマンドで定義すること。
every path	すべての path
every circle	すべての circle
every to	すべての to。なお -- には適用されない。
every node	すべての node
every rectangle node	形状が rectangle であるすべての node
every label	label オプションで作成されたすべての node

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

2.8.　数学エンジン編

この節では、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。

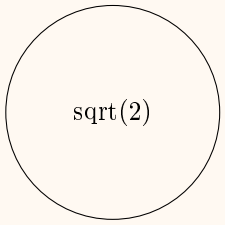
2.8.1.　マクロの定義

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が使えるとたいへん便利です。TEX にはマクロという概念がありますが、これをうまく扱うことで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` を使います。

マクロ定義

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



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 $\sqrt{2} = 1.4142\cdots$ の円が描画されています。しかしラベルを見てください。`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

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

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言えます。また置換することをマクロを展開すると呼びます。

以下の場合はエラーになります。

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

半径に単位 cm をつけたかったのですが、展開すると

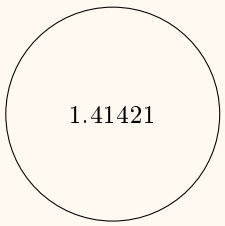
```
\draw (0,0) circle[radius=sqrt(2) cm] node {sqrt(2)};
```

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 場合を、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

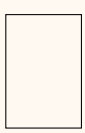
```
\begin{tikzpicture}
    \pgfmathsetmacro\r{sqrt(2)} % \r に√2 を代入
    \draw (0,0) circle[radius=\r cm] node {\r};
\end{tikzpicture}
```



ラベルには sqrt(2) の評価後の値が表示されていることが分かります。

長さのマクロ

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



上の例は \pgfmathsetlengthmacro 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。またマクロ \b の長さ（ポイント）は 1.5cm 相当であることも分かります。

\pgfmathsetmacro と \pgfmathsetlengthmacro 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

数値のマクロ 2

```
\pgfmathsetmacro{\a}{-1}
\pgfmathsetmacro{\b}{\a^2}
\pgfmathsetmacro{\c}{(\a)^2}

$a= \a, b= \b, c= \c$
```

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

を見てください。\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。

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。この構文自体は TEX や PGF のものであり、Ti*k*Z とは無関係だからです。

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。

math によるマクロ定義

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

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

\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 コマンドの引数の中では、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。

2.8.2.　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。

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

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。

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

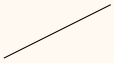
```
\begin{tikzpicture}
    \pgfmathsetmacro\x{1}
    \pgfmathsetmacro\y{2}
    \pgfmathsetmacro\z{e^\x + e^\y}
    \node (0,0) {$e^\x + e^\y = \z$};
\end{tikzpicture}
```

$e^1 + e^2 = 10.1073$

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

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



2.9.　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。

この節では、制御構文を解説します。

2.9.1.　繰り返し制御

同じコマンドを繰り返し処理したい場合は \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

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

この例では \P に値 A、B、C を代入しながら、その都度 \fill コマンドを実行しています。結果として次と同じになります。

```
\begin{tikzpicture}
    \fill (A) circle[radius=2pt];
    \fill (B) circle[radius=2pt];
    \fill (C) circle[radius=2pt];
\end{tikzpicture}
```

一般には \foreach **変数名** in {**リスト**} {**コマンド**} の構文を取ります。変数名は \ で始めます。\rad など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 \P と記述された箇所がそれらの要素に置換されます。

上の例では変数にはノード名を代入させています。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

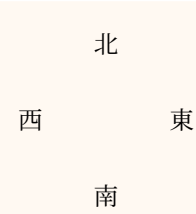
```
\begin{tikzpicture}
    \foreach \P in {(0,0),(1,0),(2,0)} \fill \P circle[radius=2pt];
\end{tikzpicture}
```

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変数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

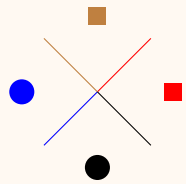
```
\begin{tikzpicture}
    \foreach \angle/\str in {0/東, 180/西, 90/北, 270/南} \node at (\angle:1)
    {\str};
\end{tikzpicture}
```



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

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

要素の置換はまさにそのままの置換なので、例えばノード名の命名などにも使えます。

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

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

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

{p, ..., 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

```
\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 コマンドの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。