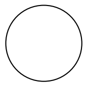
Ti*k*Z

KOVÁCS KRISTÓF, MAGYAR ANDRÁS, SIMON ANDRÁS

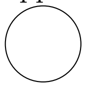
Contents

1. The simplest drawings

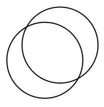
First of all, put `\usepackage{tikz}` in the preamble.

1.1. **Circles, lines,. . .** Simple things, like this inline circle  are easy to draw, just write

```
\begin{tikzpicture}
 \draw (0,0) circle (0.5);
\end{tikzpicture}
```

What is says is: draw a circle with $(0, 0)$ as its center with diameter 0.5 cm. It doesn't matter where the centre is now that there is nothing else in the drawing, because only the part of the "paper" where something is drawn appears. So there's no way to distinguish the previous drawing from this  even though this was drawn by

```
\begin{tikzpicture}
 \draw (0.2,0.2) circle (0.5);
\end{tikzpicture}
```

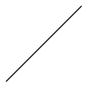
But of course, if the two circles are in the same drawing,  then the (relative) positions of the centres do matter:

```
\begin{tikzpicture}
\draw (0,0) circle (0. 5);
\draw (0.2,0.2) circle (0.5);
\end{tikzpicture}
```

Typically, we do not draw inline figures, but (floating) pictures, which therefore have a "skeleton" that looks like this:

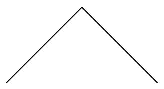
```
\begin{figure}[h]
 \begin{center}
  \begin{tikzpicture}
    ...
  \end{tikzpicture}
 \end{center}
\caption{Your caption}
\end{figure}
```

We already know how to draw a circle; here's how to draw a line segment:



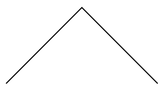
```
\begin{tikzpicture}
 \draw (0,0) -- (1,1) ;
\end{tikzpicture}
```

A broken line (?) can be drawn line segment by line segment:



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

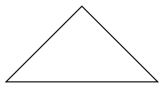
or in one go:



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

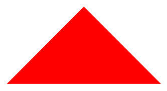
This one-shot thing (actually, what is between `\draw` and the semicolon) is called a *path*, and we'll see that it can contain more than just line segments.

One can close a path without having to specify its starting point again:



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

One advantage of this is that Ti*k*Z knows that the path is closed, which is useful for example if we want to fill the domain it borders:

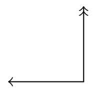


```
\begin{tikzpicture}
 \draw[fill, color=red]
   (0,0) -- (1,1) -- (2,0) -- cycle;
\end{tikzpicture}
```

We could have gotten same result with

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

We can put arrows at the ends of paths:



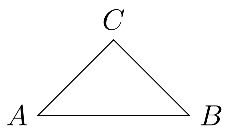
```
\begin{tikzpicture}
 \draw[<->>] (0,0) -- (1,0) -- (1,1);
\end{tikzpicture}
```

It would be easy to draw a rectangle as a closed broken line (?), but it can be done simply like this:

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

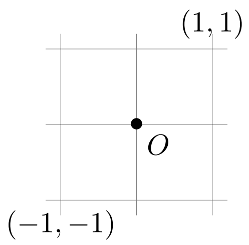
The two arguments are the coordinates of two opposite vertices.

1.2. **Naming the parts.** To label the different parts of the figure, we create a "node", which is essentially a box in which we can write:



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

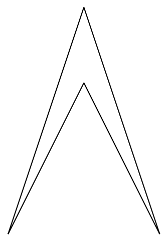
A node can be created anywhere in the path. Their position can be controlled by optional arguments. Here's another example, which also has a point with two nodes:



```
\begin{tikzpicture}
 \draw[very thin, gray]
   (-1.2,-1.2) grid (1.2,1.2);
 \draw (0,0)
   node[below right] {$O$}
   node {$\bullet$};
 \draw (-1,-1) node[below] {$(-1,-1)$};
 \draw (1,1) node[above] {$(1,1)$};
\end{tikzpicture}
```

`\draw (x, y) grid (x', y')` draws the intersection of the rectangle whose lower left corner is (x, y) and upper right corner is (x', y'), and the grid whose lines intersect each other in points with integer coordinates.

Not only can we name the parts of interest in the figure, we can also name the coordinates when drawing:

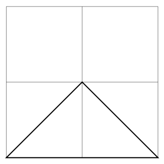


```
\begin{tikzpicture}
  \coordinate (O) at (0,0) ;
  \coordinate (O’) at (2,0);
  \coordinate (A) at (1,2) ;
  \coordinate (B) at (1,3) ;
  \draw (O) -- (A) (B) -- (O);
  \draw (O’) -- (A) (B) -- (O’) ;
\end{tikzpicture}
```

Another new feature here is that we “picked up the pen” while drawing, between points (A) and (B). (Although it would have been simpler to draw this without raising the pen.) Apart from making the code more readable, it also makes it easier to change, because we may only need to redefine one of the names, rather than going through the code changing the coordinates one by one.

1.3. **Polar coordinates.**

As we have already seen, we can work in the usual Descartes coordinate system. But one can also use the polar coordinate system (in which case the reference point is the point $(0,0)$ and the reference direction is the vector $(1,0)$). The syntax is `(angle:distance)`, where `angle` is given in degrees. The point `(1,0)` in polar coordinates is `(0:1)`.

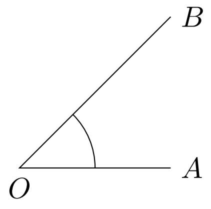


```
\begin{tikzpicture}
 \draw[very thin, gray] (0,0) grid (2,2) ;
 \draw (0,0) -- (45:{sqrt(2)})
   -- (2,0) -- cycle ;
\end{tikzpicture}
```

See §3.3 for the list of functions one can use. Here `sqrt(2)` must be enclosed in braces because it contains (normal) parentheses.

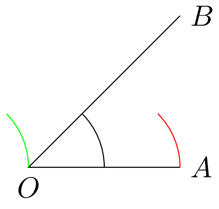
1.4. **Arcs.**

Arcs can be drawn with `arc`, but beware of a potential surprise: the point preceding it is not the center but the starting point of the arc.



```
\begin{tikzpicture}
 \draw (2,0) node[right] {$A$}
   -- (0,0) node[below] {$O$}
   -- (2,2) node[right] {$B$};
 \draw (1,0) arc (0:45:1) ;
\end{tikzpicture}
```

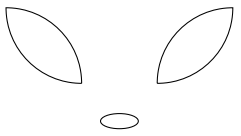
Further illustration of this:



```
\begin{tikzpicture}
 \draw (2,0) node[right] {$A$}
  -- (0,0) node[below] {$O$}
  -- (2,2) node[right] {$B$};
 \draw[green] (0,0) arc (0:45:1) ;
 \draw (1,0) arc (0:45:1) ;
 \draw[red] (2,0) arc (0:45:1) ;
\end{tikzpicture}
```

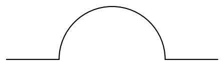
The syntax is `(b:e:r)`, where [`b,e`] is the interval of angles and `r` is the radius.

Of course, an arc may be part of a bigger path[1]:



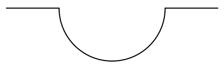
```
\begin{tikzpicture}
 \draw (0,0) arc (0:90:1) arc (180:270:1);
 \draw (1,0) arc (270:360:1) arc (90:180:1);
 \draw (0.5,-0.5)
   circle[x radius=0.25, y radius=0.1];
\end{tikzpicture}
```

which may of course contain a mixture of arcs and line sections (and grids and circles, etc.):



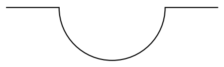
```
\begin{tikzpicture}
 \draw (0,0) -- (1,0) arc (180:0:1) -- (4,0);
\end{tikzpicture}
```

It's because of $0 - 180 < 0$ that of the two possible semi-circle the one traversed in the negative direction is drawn. If we want to draw the other arc, we need to specify the interval of angles so that its end is bigger than its beginning:



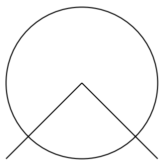
```
\begin{tikzpicture}
 \draw (0,0) -- (1,0) arc (180:360:1) -- (4,0);
\end{tikzpicture}
```

or even



```
\begin{tikzpicture}
 \draw (0,0) -- (1,0) arc (-180:0:1) -- (4,0);
\end{tikzpicture}
```

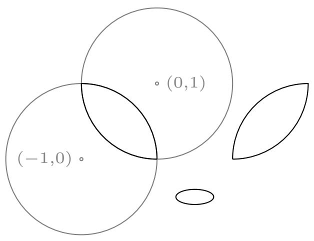
Speaking of circles as parts of a path:



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

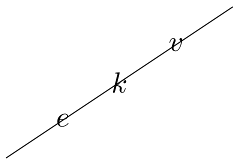
This drawing shows that the "current point" is the center of the circle after drawing it, unlike with line sections and arcs.

[1]Here's a little help for understanding what is going on:



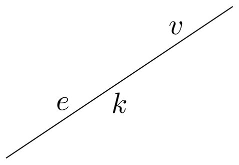
2. Phase Two

2.1. **Advanced Labeling.** We can also label the segments of a path (for example, a line section), but this is necessarily a bit more complicated. First, we need to say (after the definition of the segment) whether the label should be at its the beginning (`near start`), middle (`midway`), or end (`near end`):



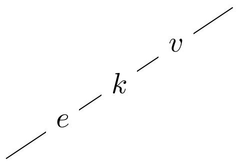
```
\begin{tikzpicture}
 \draw (0,0) -- (3,2)
   node[near start] {$e$}
   node[midway] {$k$}
   node[near end] {$v$};
\end{tikzpicture}
```

If we don't want to write *on* the line but above or below it, we can say so:



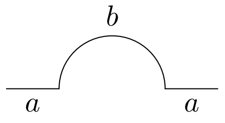
```
\begin{tikzpicture}
 \draw (0,0) -- (3,2)
  node[near start, above] {$e$}
  node[midway, below] {$k$}
  node[near end, above] {$v$};
\end{tikzpicture}
```

If we do but don't want the line to obscure the label, we paint the background white:



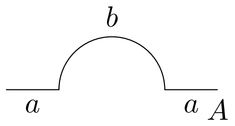
```
\begin{tikzpicture}
 \draw (0,0) -- (3,2)
   node[near start,fill=white] {$e$}
   node[midway,fill=white] {$k$}
   node[near end,fill=white] {$v$};
\end{tikzpicture}
```

In the examples above, we put multiple labels on a segment; here's an example with multiple segments with one label each:



```
\begin{tikzpicture}
  \draw (0,0) -- (1,0) node[midway,below] {$a$}
  arc (180:0:1) node[midway,above] {$b$}
  -- (4,0) node[midway,below] {$a$};
\end{tikzpicture}
```

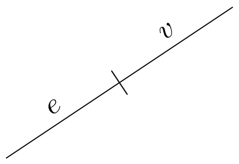
Ti*k*Z concludes from the presence of the optional arguments `near start` &c. that the node belongs to the segment and not to its endpoint:



```
\begin{tikzpicture}
  \draw (0,0) -- (1,0) node[midway,below] {$a$}
  arc (180:0:1) node[midway,above] {$b$}
  -- (4,0) node[midway,below] {$a$} node[below] {$A$};
\end{tikzpicture}
```

Here one can freely swap the two node definitions in the last line.

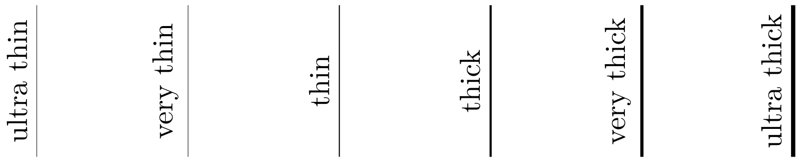
We can also write in parallel with a line segment using the `sloped` option:



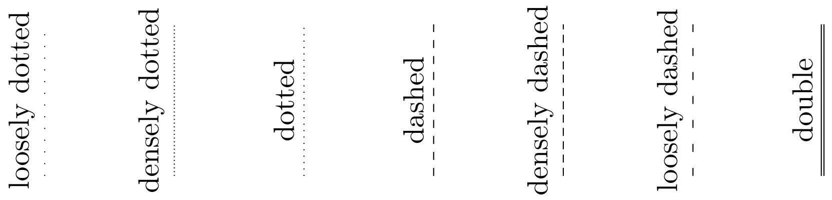
```
\begin{tikzpicture}
 \draw (0,0) -- (3,2)
   node[near start, above, sloped] {$e$}
   node[midway,sloped] {$|$}
   node[near end, above, sloped] {$v$};
\end{tikzpicture}
```

2.2. **Aesthetics.**

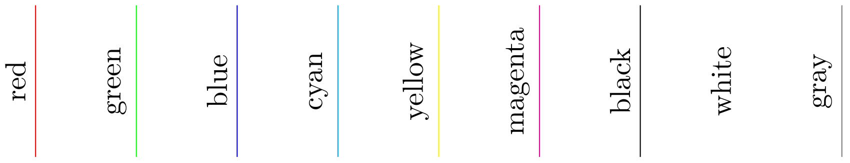
Optional arguments: line thickness. `\draw[thickness] ...`, where `thickness` is one of the following 6:



Optional arguments: line type. `\draw[linetype] ...`, where `linetype` is one of the following 7:

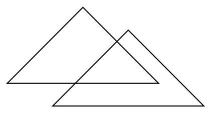


Optional arguments: colour. `\draw[color] ...`, where `color` is one of the following 9:



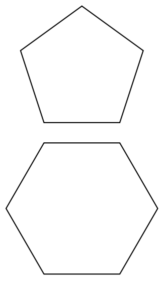
These colours can be mixed like this: `green!30!blue` (that's 30% green, 70% blue (🟦) if the second colour is missing, it defaults to white.

2.3. **Relative coordinates.** Instead of "absolute coordinates", it is often more convenient to specify a deviation from the previous point. For example, in the following drawing, thanks to the use of relative coordinates, the definition of the path defining the triangle needs to be changed in only one place.



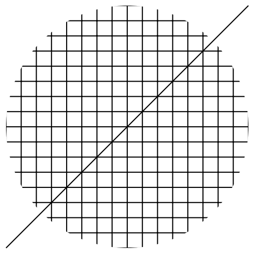
```
\begin{tikzpicture}
 \coordinate (A) at (0,0) ;
 \coordinate (A') at (0.6,-0.3) ;
 \draw (A) -- ++(1,1) -- ++(1,-1) -- cycle;
 \draw (A') -- ++(1,1) -- ++(1,-1) -- cycle;
\end{tikzpicture}
```

But it is also easier to draw a regular polygon with relative (polar) coordinates:



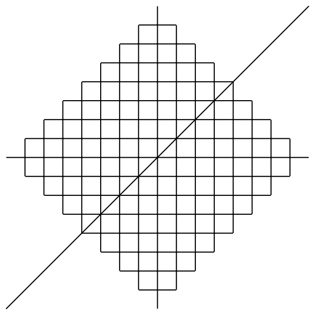
```
\begin{tikzpicture}
 \draw (0,2) -- ++(1,0) -- ++(72:1)
   -- ++(144:1) -- ++(216:1) -- cycle;
 \draw (0,0) -- ++(1,0) -- ++(60:1)
   -- ++(120:1) -- ++(180:1) --
   ++(240:1) -- ++(300:1) -- cycle;
\end{tikzpicture}
```

2.4. **Cropping.** The effect of drawing a closed path with `\clip` is to hide the part of all that is drawn *later* which lies outside the region bounded by this path. Here the closed path is a circle, so all we can see from the grid drawn later is what's inside it:



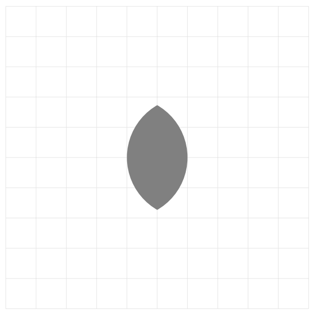
```
\begin{tikzpicture}
 \draw (-2,-2) -- (2,2);
 \clip (0,0) circle (2);
 \draw (-3,-3) grid[step=0.25] (3,3);
\end{tikzpicture}
```

Here's the same with a diamond instead of a circle:



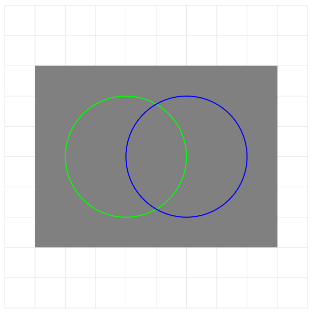
```
\begin{tikzpicture}
 \draw (-2,-2) -- (2,2);
 \clip (-2,0) -- ++(2,2) -- ++(2,-2)
   -- ++(-2,-2) -- cycle;
 \draw (-3,-3) grid[step=0.25] (3,3);
\end{tikzpicture}
```

Of course we can also cut out something from the cutout:



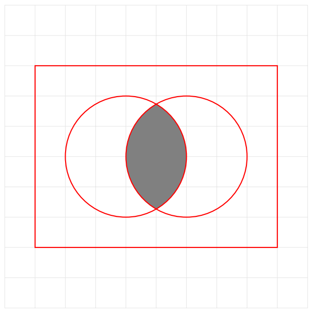
```
\begin{tikzpicture}
 \draw[gray!20,very thin]
   (-5,-5) grid (5,5);
 \clip (-1,0) circle (2);
 \clip ( 1,0) circle (2);
 \fill[gray] (-4,-3) rectangle (4,3);
\end{tikzpicture}
```

Here are the components of the previous drawing with no clipping:



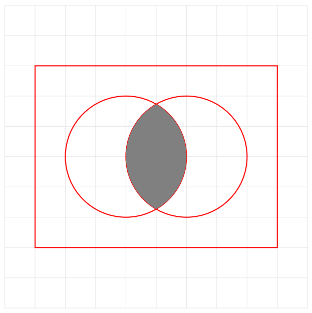
```
\begin{tikzpicture}
    \draw[gray!20,very thin] (-5,-5) grid (5,5);
    \fill[gray] (-4,-3) rectangle (4,3);
    \draw[green] (-1,0) circle (2);
    \draw[blue] ( 1,0) circle (2);
\end{tikzpicture}
```

If `\clip` is in a `scope` environment, then its effect is restricted to this environment:



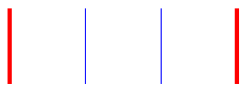
```
\begin{tikzpicture}
 \draw[gray!20,very thin]
   (-5,-5) grid (5,5);
 \begin{scope}
  \clip (-1,0) circle (2);
  \clip ( 1,0) circle (2);
  \fill[gray] (-4,-3) rectangle (4,3);
 \end{scope}
 \draw[red] (-4,-3) rectangle (4,3);
 \draw[red] (-1,0) circle (2);
 \draw[red] (1,0) circle (2);
\end{tikzpicture}
```

Here we could have achieved the same effect without `scope` by changing the order:



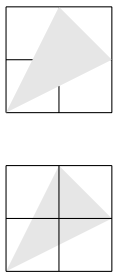
```
\begin{tikzpicture}
 \draw[gray!20,very thin]
   (-5,-5) grid (5,5);
 \draw[red] (-4,-3) rectangle (4,3);
 \draw[red] (-1,0) circle (2);
 \draw[red] (1,0) circle (2);
 \clip (-1,0) circle (2);
 \clip ( 1,0) circle (2);
 \fill[gray] (-4,-3) rectangle (4,3);
\end{tikzpicture}
```

But varying the order is not always feasible, because what is drawn later obscures what is already on the paper. This will be discussed in more detail in the next section. In the meantime: here's another application of `scope`:



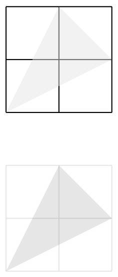
```
\begin{tikzpicture}[ultra thick,red]
 \draw (0,0) -- (0,1);
 \begin{scope}[thin,blue]
  \draw (1,0) -- (1,1);
  \draw (2,0) -- (2,1);
 \end{scope}
 \draw (3,0) -- (3,1);
\end{tikzpicture}
```

2.5. **Transparency.** An example showing that the order of drawings counts:



```
\begin{tikzpicture}
 \draw (-1,2) grid ++(2,2) ;
 \fill[gray!20] (-1,2) -- ++(1,2)
   -- ++(1,-1) -- cycle ;
 \fill[gray!20] (-1,-1) -- ++(1,2)
   -- ++(1,-1) -- cycle ;
 \draw (-1,-1) grid ++(2,2) ;
\end{tikzpicture}
```

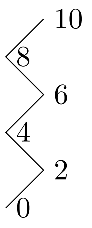
So even light gray obscured black. In fact, white would also have obscured it, and we would have gotten this: , we chose gray only for the sake of symmetry, because in the bottom figure the white triangle would have been just as spectacular as the white line on a white background was a little earlier. If we want the thing we drew later to not obscure what we drew earlier, we can use the optional parameter `opacity`. Its value is a number between 0 (completely transparent) and 1 (completely opaque, which is the default).



```
\begin{tikzpicture}
 \draw (-1,2) grid ++(2,2) ;
 \fill[gray!20,opacity=0.5] (-1,2)
  -- ++(1,2) -- ++(1,-1) -- cycle ;
 \fill[gray!20] (-1,-1)
  -- ++(1,2) -- ++(1,-1) -- cycle ;
 \draw[opacity=0.1] (-1,-1) grid ++(2,2) ;
\end{tikzpicture}
```

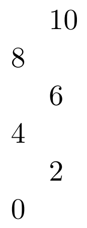
3. Advanced

3.1. **Loops.**



```
\begin{tikzpicture}
 \draw (0,0) foreach \x in {0,2,...,10}
  { -- ({mod(\x,4)/4},\x/4) node[right] {\x}};
\end{tikzpicture}
```

`mod(\x,4)/4` must be enclosed in braces because it contains (normal) parentheses. `foreach` here was a “operation” (the official term is *path extension operation*) just like `--` or `arc` or `rectangle`. Hence outside of a path it doesn't make sense, but there is a *command* `\foreach` that one can use:

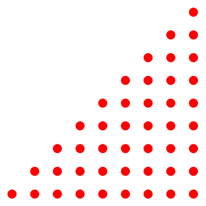


```
\begin{tikzpicture}
 \foreach \x in {0,2,...,10} \draw
  ({mod(\x,4)/4},\x/4) node[right] {\x};
\end{tikzpicture}
```

The `\foreach` command can in fact be used anywhere, even outside the `tikzpicture` environment. An example is $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, which was written like this:

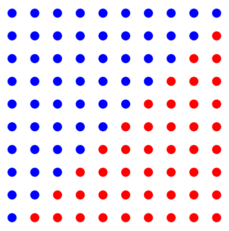
```
\foreach \x in {1,2,3} {$x_\x=\x$, }
```

Loops can be nested:



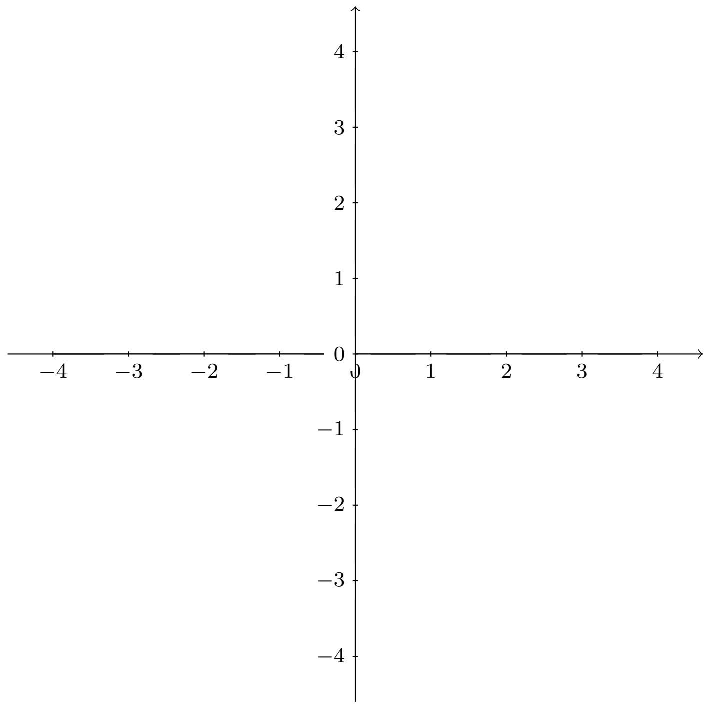
```
\begin{tikzpicture}
 \foreach \x in {1,...,10}{
  \foreach \y in {1,...,10}{
   \ifnum \x>\y
    \fill[red] (\x,\y) circle (0.2);
   \fi
  }
 }
\end{tikzpicture}
```

`\ifnum ... \fi` (in all its glory: `\ifnum ... \else ... \fi`) is not a Ti*k*Z, but a LaTeX, and in fact, a TeXcommand. An example of using `\else`:



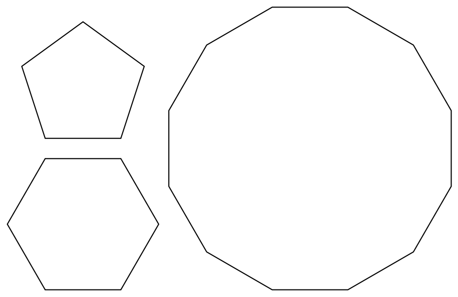
```
\begin{tikzpicture}
 \foreach \x in {1,...,10}{
  \foreach \y in {1,...,10}{
   \ifnum \x>\y
    \fill[red] (\x,\y) circle (0.2);
   \else
    \fill[blue] (\x,\y) circle (0.2);
   \fi
  }
 }
\end{tikzpicture}
```

When drawing a coordinate system, it is very natural to use loops:



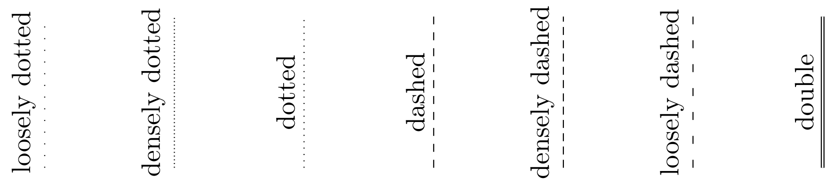
```
\draw (-4,0) node[below] {$\scriptstyle-4$}
  foreach \i in {-3,-2,...,4} {-- (\i,0)
    node[below,fill=white] {$\scriptstyle\i$}} ;
\foreach \i in {-4,-3,...,4} \draw (\i,-1pt) -- (\i,1pt);
\draw[->] (-4.6,0) -- (4.6,0);
\draw (0,-4) node[left] {$\scriptstyle-4$}
  foreach \i in {-3,-2,...,4} {-- (0,\i)
    node[left,fill=white] {$\scriptstyle\i$}} ;
\foreach \i in {-4,-3,...,4} \draw (-1pt,\i) -- (1pt,\i);
\draw[->] (0,-4.6) -- (0,4.6);
```

The same goes for drawing regular polygons:



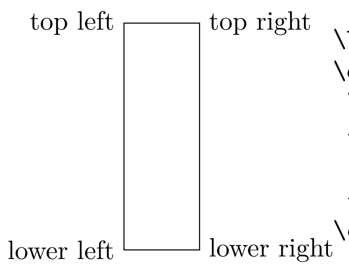
```
\begin{tikzpicture}
  \draw (0,2)
    foreach \i in {0,...,3}
     {-- ++(72*\i:1)} -- cycle;
  \draw (0,0)
    foreach \i in {0,...,4}
     {-- ++(60*\i:1)} -- cycle;
  \draw (3,0)
    foreach \i in {0,...,10}
     {-- ++(30*\i:1)} -- cycle;
\end{tikzpicture}
```

The figure way above showing line types was also made with `\foreach`:



```
\begin{tikzpicture}
 \foreach \x / \y in
   {loosely dotted/1,densely dotted/2,dotted/3,dashed/4,
     densely dashed/5,loosely dashed/6,double/7}
     { \draw[\x] (12*\y/7,0) --  ++(0,2)
        node[midway,above,sloped] {\x};}
\end{tikzpicture}
```

What is new here is that the list consists of ordered pairs where the members of the pair (and the variables running over them) are separated by /. One can similarly loop over a list of ordered triples, quadruplets, etc.:

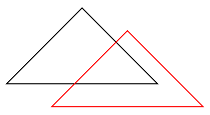


```
\begin{tikzpicture}
\draw (0,0) rectangle (1,3);
 \foreach \x / \y / \label / \where in
 {0/0/lower left/left, 1/0/lower right/right,
  1/3/top right/right, 0/3/top left/left}
 {\draw (\x, \y) node[\where]{\label}; }
\end{tikzpicture}
```

The same with the *path extension operation* foreach instead of the command \foreach:

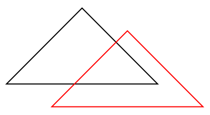
```
\begin{tikzpicture}
\draw (0,) foreach \x / \y / \label / \where in
 {0/0/lower left/left, 1/0/lower right/right,
  1/3/top right/right, 0/3/top left/left}
 { -- (\x, \y) node[\where]{\label} } -- cycle ;
\end{tikzpicture}
```

3.2. **Transformations.** Previously, a figure like the following was used to illustrate relative coordinates:



```
\begin{tikzpicture}
 \draw (0,0) -- ++(1,1)
   -- ++(1, -1) -- cycle;
 \draw[red] (0.6, -0.3) -- ++(1,1)
   -- ++(1, -1) -- cycle;
\end{tikzpicture}
```

The same effect can be achieved by drawing the same triangle shifted:



```
\begin{tikzpicture}
\draw (0,0) -- ++(1,1) -- ++(1,-1) -- cycle;
\draw[red,shift ={(0.6,-0.3)}] (0,0) -
  ++(1,1) -- ++(1,-1) -- cycle;
\end{tikzpicture}
```

This could have been written like this:

```
\draw (0,0) -- ++(1,1) -- ++(1,-1) -- cycle
  [shift ={(0.6,-0.3)}] (0,0) -- ++(1,1) -- ++(1,-1) -- cycle;
```

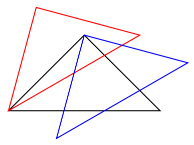
that is, we can apply shift (or more generally: a transformation) in the middle of a path .

Technical remark: in the original example, we named the coordinates of the lower left corners, but here

```
\begin{tikzpicture}
\coordinate (A) at (0,0);
\draw (A) -- ++(1,1) -- ++(1,-1) -- cycle;
\draw[red,shift ={(0.6,-0.3)}] (A) --
  ++(1,1) -- ++(1,-1) -- cycle;
\end{tikzpicture}
```

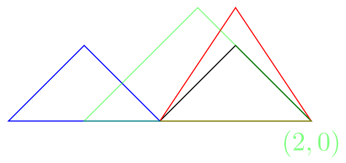
would not work because `(A)` stores the actual point whose coordinates are `(0,0)` (before applying the shift), and not the coordinates. Try it!

Besides shifting, there is also rotation (`rotate`):



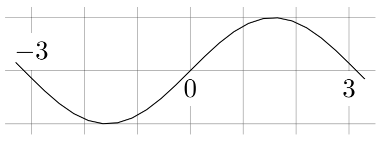
```
\begin{tikzpicture}
\draw (0,0) -- ++(1,1) -- ++(1,-1) -- cycle;
\draw[red,rotate=30] (0,0) --
  ++(1,1) -- ++(1,-1) -- cycle;
\draw[blue,rotate around={30: (1,1)}]
  (0,0) -- ++(1,1) -- ++(1,-1) -- cycle;
\end{tikzpicture}
```

scaling (`scale`), also separately along the axes (`xscale`, `yscale`):



```
\begin{tikzpicture}
 \draw (0,0) -- ++(1,1)
   -- ++(1,-1) -- cycle;
 \draw[red,yscale=1.5] (0,0) --
   ++(1,1) -- ++(1,-1) -- cycle;
 \draw[blue,xscale=-1] (0,0) --
   ++(1,1) -- ++(1,-1) -- cycle;
 \draw [opacity=0.5,green,
   scale around={1.5:(2,0)}]
   (0,0) -- ++(1,1) -- ++(1,-1)
   node[below] {$(2,0)$} -- cycle;
\end{tikzpicture}
```

3.3. **Plotting functions.**



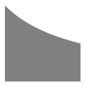
```
\begin{tikzpicture}
 \draw[very thin,gray]
  (-3.5,-1.2) grid (3.5,1.2);
 \draw (0,0)
  node[below,fill=white]{$0$};
 \draw (3,0)
  node[below,fill=white]{$3$};
 \draw (-3.0)
  node[above,fill=white]{$-3$};
 \draw plot[domain=-3.3:3.3]
  (\x,{sin(deg(\x))});
\end{tikzpicture}
```

Available functions:

- `+`, `-`, `*`, `/`
- `mod`, `min`, `max`
- `abs`, `exp`, `ln`, `sqrt`
- `round`, `floor`, `ceil`
- `sin`, `cos`, `tan`
- `asin`, `acos`, `atan`
- `pi`, `deg`, `rad`
- `rnd` (random number between 0 and 1), `rand` (random number between −1 to 1)
- `==`, `<`, `>` (result is 0 if false, 1 if true)

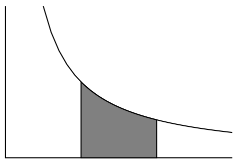
Trigonometric functions expect their argument in degrees, that's why we wrote `sin(deg(\x))` in the drawing above. This is such a common problem that one can write the same thing as `sin(\x r)`.

`plot`, like all path extension operations, can be part of a path (in the example above it was also part of a path, it's just that it was the *only* part):



```
\fill[gray] (1,0) -- (1,1) --
  plot[domain=1:2] (\x,1/\x) -- (2,0) -- cycle;
```

This is useful, e.g. in a figure like this:



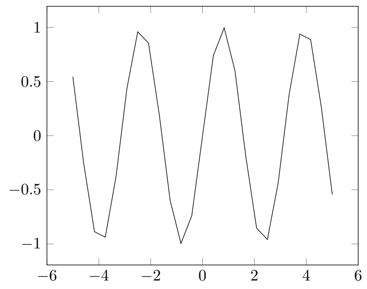
```
\draw (3,0) -- (0,0) -- (0,2);
\draw plot[domain=0.5:3] (\x,1/\x);
\fill[gray,draw=black] (1,0) -- (1,1) --
  plot[domain=1:2] (\x,1/\x) -- (2,0) -- cycle;
```

Appendix A. How to *really* plot functions

We have already seen how to make the graph of a function part of a *path*. Most of the time, however, we do not want this, but we want to plot a function in some coordinate system, including not having to draw the coordinate axes ourselves. In this case, it is better to use the `pgfplots` package. To do this, put the following in the preamble:

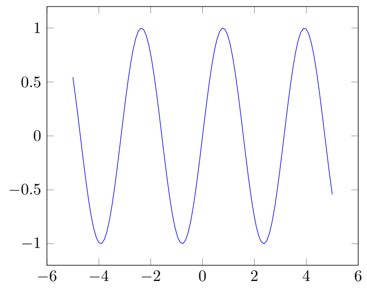
```
\usepackage{pgfplots}
\pgfplotsset{compat = newest}
```

As an example, let's plot the function $\sin 2x$ on the interval $[-5, 5]$!



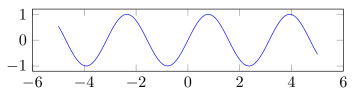
```
\begin{tikzpicture}
 \begin{axis}[domain=-5:5]
 \addplot []
   {sin(deg(2*x))};
 \end{axis}
\end{tikzpicture}
```

This is a bit rough, but we can improve it with the `samples` parameter, which tells `pgfplots` how many points to compute the function values on the interval, which are then connected.



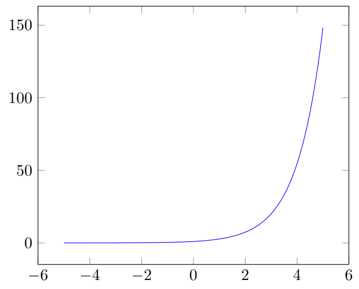
```
\begin{tikzpicture}
 \begin{axis}[]
 \addplot [blue, domain=-5:5, samples=100]
   {sin(deg(2*x))};
 \end{axis}
\end{tikzpicture}
```

If we're bothered by the unit being different on the two axes (i.e., that the *aspect ratio* is not 1), we can do this:



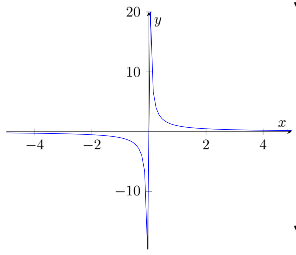
```
\begin{tikzpicture}
 \begin{axis}[axis equal image=true]
 %             ^^^ aspect ratio = 1
 \addplot [blue, domain=-5:5, samples=100]
   {sin(deg(2*x))};
 \end{axis}
\end{tikzpicture}
```

The automatic *aspect ratio* can come in handy, e.g. here:



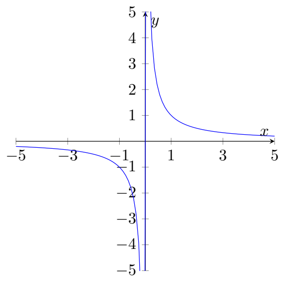
```
\begin{tikzpicture}
 \begin{axis}[]
 \addplot [blue, domain=-5:5, samples=100]
   {exp(x)};
 \end{axis}
\end{tikzpicture}
```

But it does not always help:



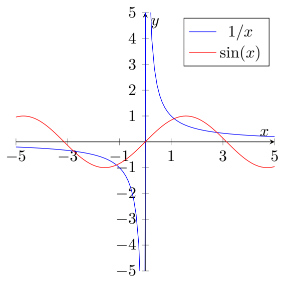
```
\begin{tikzpicture}
 \begin{axis}[axis x line=center,
  axis y line=center,
  xlabel={$x$}, ylabel={$y$}]
 \addplot [blue, domain=-5:5, samples=100]
   {1/x))};
 \end{axis}
\end{tikzpicture}
```

(By the way, we switched to the usual way of drawing coordinate axes.) Here we are better off clipping the outlying values: this is what `ymin` and `ymax` are for.



```
\begin{tikzpicture}
 \begin{axis}[axis equal image=true,
 ymax=5, ymin=-5, axis x line=center,
 axis y line=center, xtick={-5,-3,...,5},
 ytick={-5,-4,...,5}, xlabel={$x$}, ylabel={$y$}]
 \addplot [blue, domain=-5:5, samples=100]
   {1/x))};
 \end{axis}
\end{tikzpicture}
```

Finally: we can plot several functions at once. In this case, it is worth using more colors, and with the `\legend` command tell which color belongs to which function.



```
\begin{tikzpicture}
 \begin{axis}[axis equal image=true,
 ymax=5, ymin=-5, axis x line=center,
 axis y line=center, xtick={-5,-3,...,5},
 ytick={-5,-4,...,5}, xlabel={$x$}, ylabel={$y$}]
 \addplot [blue, domain=-5:5, samples=100]
   {1/x))};
 \addplot[red, domain=-5:5, samples=100]
   {sin(deg(x))};
\legend{$1/x$,$\sin(x)$}
 \end{axis}
\end{tikzpicture}
```