2 Tutorial: A Picture for Karl's Students
教程：为卡尔的学生绘制图片

This tutorial is intended for new users of Ti*k*Z. It does not give an exhaustive account of all the features of Ti*k*Z, just of those that you are likely to use right away.

本教程适用于初次使用 Ti*k*Z 的用户。它并不详尽地介绍 Ti*k*Z 的所有功能，只介绍了你可能会立即使用的部分。

Karl is a math and chemistry high-school teacher. He used to create the graphics in his worksheets and exams using LaTeX's `{picture}` environment. While the results were acceptable, creating the graphics often turned out to be a lengthy process. Also, there tended to be problems with lines having slightly wrong angles and circles also seemed to be hard to get right. Naturally, his students could not care less whether the lines had the exact right angles and they find Karl's exams too difficult no matter how nicely they were drawn. But Karl was never entirely satisfied with the result.

卡尔是一位数学和化学高中教师。他过去使用 LaTeX 的 `{picture}` 环境在他的教学材料和考试中创建图形。虽然结果还可以接受，但创建图形往往是一个耗时的过程。而且，直线的角度可能稍有偏差，画圆也似乎很难做到完美。自然地，他的学生并不关心线条是否具有准确的角度，无论图形画得多好看，他们都觉得卡尔的考试太难了。但是卡尔对结果从未完全满意。

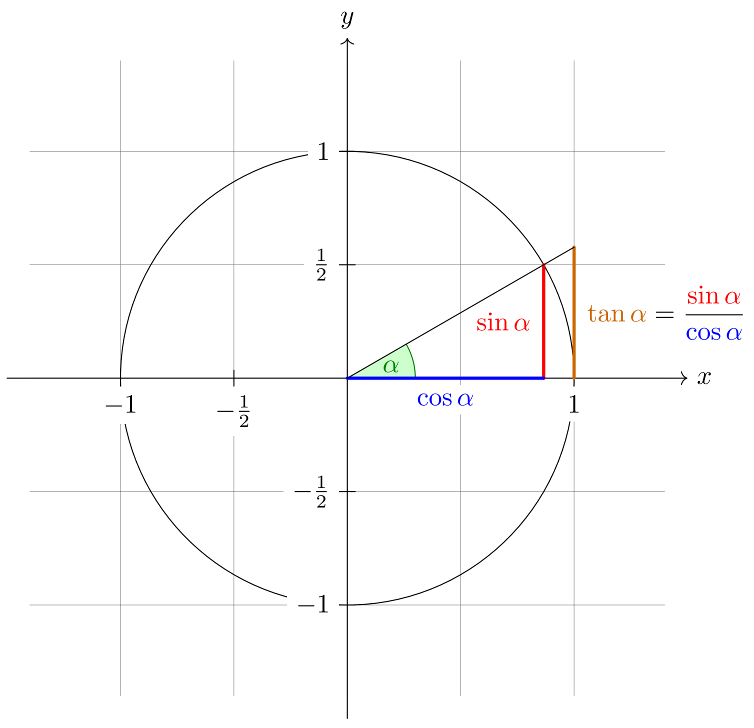
Karl's son, who was even less satisfied with the results (he did not have to take the exams, after all), told Karl that he might wish to try out a new package for creating graphics. A bit confusingly, this package seems to have two names: First, Karl had to download and install a package called PGF. Then it turns out that inside this package there is another package called Ti*k*Z, which is supposed to stand for "Ti*k*Z ist *kein* Zeichenprogramm". Karl finds this all a bit strange and Ti*k*Z seems to indicate that the package does not do what he needs. However, having used GNU software for quite some time and "GNU not being Unix", there seems to be hope yet. His son assures him that Ti*k*Z's name is intended to warn people that Ti*k*Z is not a program that you can use to draw graphics with your mouse or tablet. Rather, it is more like a "graphics language".

卡尔的儿子对结果更不满意（毕竟他不必参加考试），他告诉卡尔可以尝试一个新的用于创建图形的软件包。有点令人困惑的是，这个软件包似乎有两个名称：首先，卡尔需要下载和安装一个名为 PGF 的软件包。然后发现，在这个软件包中有另一个名为 Ti*k*Z 的软件包，它的缩写意味着 Ti*k*Z 不是绘图程序''。卡尔觉得这一切有点奇怪，而且 Ti*k*Z 似乎表明这个软件包并不能满足他的需求。然而，他使用 GNU 软件已经有一段时间了，而"GNU 不是 Unix''，这似乎还有希望。他的儿子向他保证，Ti*k*Z 的名字旨在告诫人们，Ti*k*Z 不是一个可以用鼠标或平板电脑绘制图形的程序。相反，它更像是一种 ''图形语言''。

2.1 Problem Statement
问题陈述

Karl wants to put a graphic on the next worksheet for his students. He is currently teaching his students about sine and cosine. What he would like to have is something that looks like this (ideally):

Karl 想在下一张工作表上为他的学生放置一个图形。他目前正在教学生正弦和余弦。他希望有一个看起来像这样的东西（理想情况下）:



The angle α is 30° in the example ($\pi/6$ in radians). The sine of α, which is the height of the red line, is
这个角度 α在例子中是 30°（弧度为 $\pi/6$）。α 的正弦，也就是红线的高度，是

$$\sin\alpha = 1/2.$$

By the Theorem of Pythagoras we have $\cos^2\alpha + \sin^2\alpha = 1$. Thus the length of the blue line, which is the cosine of α, must be
根据勾股定理，我们有 $\cos^2\alpha+\sin^2\alpha = 1$。因此，蓝线的长度，也就是$\alpha$ 的余弦，必须是

$$\cos\alpha = \sqrt{1-1/4} = \tfrac{1}{2}\sqrt{3}.$$

This shows that $\tan\alpha$, which is the height of the orange line, is
这表明α 的正切，也就是橙线的高度，是

$$\tan\alpha = \frac{\sin\alpha}{\cos\alpha} = 1/\sqrt{3}.$$

2.2 Setting up the Environment 设置环境

In Ti*k*Z, to draw a picture, at the start of the picture you need to tell TEX or LATEX that you want to start a picture. In LATEX this is done using the environment `{tikzpicture}`, in plain TEX you just use `\tikzpicture` to start the picture and `\endtikzpicture` to end it.

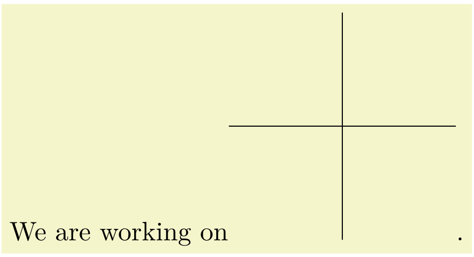
在 Ti*k*Z 中，要绘制一个图形，你需要在图形开始时告诉 TEX 或 LATEX 你想要开始一个图形。在 LATEX 中，使用环境`{tikzpicture}` 来完成这个操作，在 plain TEX 中，你只需使用`\tikzpicture` 来开始图形，使用`\endtikzpicture` 来结束图形。

2.2.1 Setting up the Environment in LATEX 在 LATEX 中设置环境

Karl, being a LATEX user, thus sets up his file as follows:
作为一个 LATEX 用户，Karl 设置他的文件如下：

```
\documentclass{article} % say
\usepackage{tikz}
\begin{document}
We are working on
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
\end{tikzpicture}.
\end{document}
```

When executed, that is, run via `pdflatex` or via `latex` followed by `dvips`, the resulting will contain something that looks like this:
当执行该代码，即通过 `pdflatex` 或 `latex` 后跟 `dvips` 运行时，结果将会包含以下内容：



```
We are working on
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
\end{tikzpicture}.
```

Admittedly, not quite the whole picture, yet, but we do have the axes established. Well, not quite, but we have the lines that make up the axes drawn. Karl suddenly has a sinking feeling that the picture is still some way off.

诚然，这还不是完整的图像，但我们已经建立了坐标轴。嗯，并不完全准确，但我们已经绘制了构成坐标轴的线段。Karl 突然有一种不安的感觉，认为图像还有一段路要走。

Let's have a more detailed look at the code. First, the package `tikz` is loaded. This package is a so-called "frontend" to the basic PGF system. The basic layer, which is also described in this manual, is somewhat more, well, basic and thus harder to use. The frontend makes things easier by providing a simpler syntax.

让我们更详细地看一下代码。首先，加载了 `tikz` 宏包。该宏包是基本 PGF 系统的所谓"前端"。基本层在本手册中也有所描述，它更基础一些，因此使用起来更困难。前端通过提供更简单的语法使事情更容易。

Inside the environment there are two `\draw` commands. They mean: "The path, which is specified following the command up to the semicolon, should be drawn." The first path is specified as `(-1.5,0) -- (0,1.5)`, which means "a straight line from the point at position $(-1.5, 0)$ to the point at position $(0, 1.5)$". Here, the positions are specified within a special coordinate system in which, initially, one unit is 1cm.

在环境内部有两个 `\draw` 命令。它们的意思是："应该绘制路径，路径在命令后面到分号之前指定。"第一个路径被指定为 `(-1.5,0) -- (0,1.5)`，意思是"从位置 $(-1.5, 0)$ 到位置 $(0, 1.5)$ 绘制一条直线"。这里，位置在一个特殊的坐标系中指定，初始情况下，一个单位等于 1cm。

Karl is quite pleased to note that the environment automatically reserves enough space to encompass the picture.

Karl 很高兴地注意到环境会自动保留足够的空间来容纳图片。

2.2.2 Setting up the Environment in Plain TeX 在 Plain TeX 中设置环境

Karl's wife Gerda, who also happens to be a math teacher, is not a LaTeX user, but uses plain TeX since she prefers to do things "the old way". She can also use Ti*k*Z. Instead of `\usepackage{tikz}` she has to write `\input tikz.tex` and instead of `\begin{tikzpicture}` she writes `\tikzpicture` and instead of `\end{tikzpicture}` she writes `\endtikzpicture`.

Karl 的妻子 Gerda 也是一名数学教师，她不使用 LaTeX，而是使用 Plain TeX，因为她更喜欢"老派"的方式。她也可以使用 Ti*k*Z。她需要将 `\usepackage{tikz}` 替换为 `\input tikz.tex`，将 `\begin{tikzpicture}` 替换为 `\tikzpicture`，将 `\end{tikzpicture}` 替换为 `\endtikzpicture`。

Thus, she would use:

因此，她将使用以下代码：

```
%% Plain TeX file
\input tikz.tex
\baselineskip=12pt
\hsize=6.3truein
\vsize=8.7truein
We are working on
\tikzpicture
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
\endtikzpicture.
\bye
```

Gerda can typeset this file using either `pdftex` or `tex` together with `dvips`. Ti*k*Z will automatically discern which driver she is using. If she wishes to use `dvipdfm` together with `tex`, she either needs to modify the file `pgf.cfg` or can write `\def\pgfsysdriver{pgfsys-dvipdfm.def}` somewhere *before* she inputs `tikz.tex` or `pgf.tex`.

Gerda 可以使用 `pdftex` 或 `tex` 与 `dvips` 来排版此文件。Ti*k*Z 会自动判断她使用的是哪个驱动程序。如果她希望使用 `dvipdfm` 与 `tex` 一起使用，她需要修改文件 `pgf.cfg`，或者可以在输入 `tikz.tex` 或 `pgf.tex` 之前的某个地方写上 `\def\pgfsysdriver{pgfsys-dvipdfm.def}`。

2.2.3 Setting up the Environment in ConTEXt
在 ConTEXt 中设置环境

Karl's uncle Hans uses ConTEXt. Like Gerda, Hans can also use Ti*k*Z. Instead of `\usepackage{tikz}` he says `\usemodule[tikz]`. Instead of `\begin{tikzpicture}` he writes `\starttikzpicture` and instead of `\end{tikzpicture}` he writes `\stoptikzpicture`.

Karl 的叔叔 Hans 使用 ConTEXt。和 Gerda 一样，Hans 也可以使用 Ti*k*Z。他需要将 `\usepackage{tikz}` 替换为 `\usemodule[tikz]`。将 `\begin{tikzpicture}` 替换为 `\starttikzpicture`，将 `\end{tikzpicture}` 替换为 `\stoptikzpicture`。

His version of the example looks like this:

他的示例代码如下：

```
%% ConTeXt file
\usemodule[tikz]

\starttext
  We are working on
  \starttikzpicture
    \draw (-1.5,0) -- (1.5,0);
    \draw (0,-1.5) -- (0,1.5);
  \stoptikzpicture.
\stoptext
```

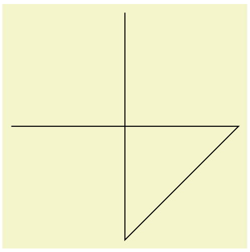
Hans will now typeset this file in the usual way using `texexec` or `context`.

Hans 可以使用 `texexec` 或 `context` 来排版这个文件。

2.3 Straight Path Construction
直线路径构建

The basic building block of all pictures in Ti*k*Z is the path. A *path* is a series of straight lines and curves that are connected (that is not the whole picture, but let us ignore the complications for the moment). You start a path by specifying the coordinates of the start position as a point in round brackets, as in `(0,0)`. This is followed by a series of "path extension operations". The simplest is `--`, which we used already. It must be followed by another coordinate and it extends the path in a straight line to this new position. For example, if we were to turn the two paths of the axes into one path, the following would result:

在 Ti*k*Z 中，所有图片的基本构建块是路径。一个*路径*是由连接的直线和曲线组成的序列（这并不构成整个图片，但我们暂且忽略这些复杂性）。你可以通过将起始位置的坐标作为圆括号中的点来开始一个路径，例如 `(0,0)`。接下来是一系列的"路径扩展操作"。最简单的操作是 `--`，我们已经使用过了。它必须后跟另一个坐标，并将路径以直线延伸到新的位置。例如，如果我们将坐标轴的两条路径合并为一条路径，结果如下：



```
\tikz \draw (-1.5,0) -- (1.5,0) -- (0,-1.5) -- (0,1.5);
```

Karl is a bit confused by the fact that there is no `{tikzpicture}` environment, here. Instead, the little command `\tikz` is used. This command either takes one argument (starting with an opening brace as in `\tikz{\draw (0,0) -- (1.5,0)}`, which yields ———) or collects everything up to the next semicolon and puts it inside a `{tikzpicture}` environment. As a rule of thumb, all Ti*k*Z graphic drawing commands must occur as an argument of `\tikz` or inside a `{tikzpicture}` environment. Fortunately, the command `\draw` will only be defined inside this environment, so there is little chance that you will accidentally do something wrong here.

卡尔有点困惑，因为这里没有 `{tikzpicture}` 环境。取而代之，使用了命令 `\tikz`。这个命令可以接受一个参数（以打开的大括号开始，比如 `\tikz{\draw (0,0) -- (1.5,0)}`，它会生成 ———）或者收集直到下一个分号的所有内容，并将其放入 `{tikzpicture}` 环境中。作为经验法则，所有的 Ti*k*Z 图形绘制命令必须作为 `\tikz` 的参数或者在 `{tikzpicture}` 环境内部。幸运的是，命令 `\draw` 只会在此环境内部定义，所以你很少会在这里意外地做错事。

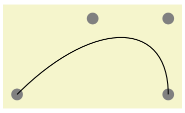
2.4 Curved Path Construction
曲线路径构建

The next thing Karl wants to do is to draw the circle. For this, straight lines obviously will not do. Instead, we need some way to draw curves. For this, Ti*k*Z provides a special syntax. One or two "control points" are needed. The math behind them is not quite trivial, but here is the basic idea: Suppose you are at point x and the first control point is y. Then the curve will start "going in the direction of y at x", that is, the tangent of the curve at x will point toward y. Next, suppose the curve should end at z and the second support point is w. Then the curve will, indeed, end at z and the tangent of the curve at point z will go through w.

卡尔接下来想要做的是绘制圆。显然，直线无法实现这一点。相反，我们需要一种方法来绘制曲线。为此，Ti*k*Z 提供了一种特殊的语法。需要一个或两个 "控制点"。它们背后的数学并不是非常复杂，但基本思想是：假设你在点 x，第一个控制点是 y。那么曲线将在 "以 x 为中心向 y 的方向开始"，也就是说，曲线在点 x 的切线将指向 y。接下来，假设曲线应该在 z 结束，第二个支撑点是 w。那么曲线确实会在 z 结束，并且曲线在点 z 的切线将经过 w。

Here is an example (the control points have been added for clarity):

这是一个示例（为了清晰起见，添加了控制点）：



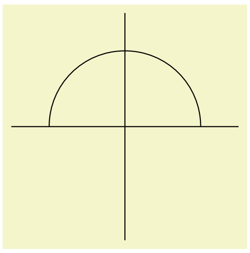
```
\begin{tikzpicture}
  \filldraw [gray] (0,0) circle [radius=2pt]
                   (1,1) circle [radius=2pt]
                   (2,1) circle [radius=2pt]
                   (2,0) circle [radius=2pt];
  \draw (0,0) .. controls (1,1) and (2,1) .. (2,0);
\end{tikzpicture}
```

The general syntax for extending a path in a "curved" way is `.. controls` ⟨*first control point*⟩ `and` ⟨*second control point*⟩ `..` ⟨*end point*⟩. You can leave out the `and` ⟨*second control point*⟩, which causes the first one to be used twice.

通过使用 `.. controls` ⟨ *第一个控制点*⟩ `and` ⟨ *第二个控制点*⟩ `..` ⟨ *结束点*⟩ 的一般语法，可以以 "曲线" 的方式扩展路径。可以省略 `and` ⟨ *第二个控制点*⟩，这样将使用第一个控制点两次。

So, Karl can now add the first half circle to the picture:

因此，卡尔现在可以将第一个半圆添加到图片中：



```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (-1,0) .. controls (-1,0.555) and (-0.555,1) .. (0,1)
               .. controls (0.555,1) and (1,0.555) .. (1,0);
\end{tikzpicture}
```

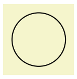
Karl is happy with the result, but finds specifying circles in this way to be extremely awkward. Fortunately, there is a much simpler way.

Karl 对结果感到满意，但他发现用这种方式指定圆非常麻烦。幸运的是，有一种更简单的方法。

2.5 Circle Path Construction
绘制圆形路径

In order to draw a circle, the path construction operation `circle` can be used. This operation is followed by a radius in brackets as in the following example: (Note that the previous position is used as the *center* of the circle.)

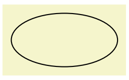
为了绘制一个圆，可以使用路径构造操作`circle`。该操作后跟在括号内的半径，如下面的示例所示：（请注意，前一个位置被用作圆的*中心*。）



```
\tikz \draw (0,0) circle [radius=10pt];
```

You can also append an ellipse to the path using the `ellipse` operation. Instead of a single radius you can specify two of them:

您还可以使用`ellipse` 操作将椭圆附加到路径上。您可以指定两个半径，而不是一个：

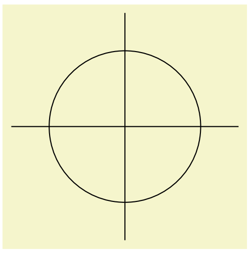


```
\tikz \draw (0,0) ellipse [x radius=20pt, y radius=10pt];
```

To draw an ellipse whose axes are not horizontal and vertical, but point in an arbitrary direction (a "turned ellipse" like) you can use transformations, which are explained later. The code for the little ellipse is `\tikz \draw[rotate=30] (0,0) ellipse [x radius=6pt, y radius=3pt];`, by the way.

要绘制一个轴不是水平和垂直的椭圆，而是指向任意方向的椭圆（一个 "旋转椭圆"，如 ），您可以使用后面将解释的变换。顺便说一下，小椭圆的代码是
`\tikz \draw[rotate=30] (0,0) ellipse [x radius=6pt, y radius=3pt];`。

So, returning to Karl's problem, he can write `\draw (0,0) circle [radius=1cm];` to draw the circle:
因此，回到 Karl 的问题，他可以写`\draw (0,0) circle [radius=1cm];` 来绘制圆：



```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

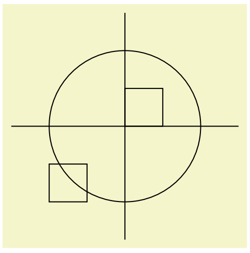
At this point, Karl is a bit alarmed that the circle is so small when he wants the final picture to be much bigger. He is pleased to learn that Ti*k*Z has powerful transformation options and scaling everything by a factor of three is very easy. But let us leave the size as it is for the moment to save some space.

此时，Karl 有点担心当他希望最终的图片更大时，圆太小了。他很高兴地知道 Ti*k*Z 有强大的变换选项，将所有内容放大三倍非常容易。但为了节省空间，让我们暂时保持尺寸不变。

2.6 Rectangle Path Construction 矩形路径构造

The next things we would like to have is the grid in the background. There are several ways to produce it. For example, one might draw lots of rectangles. Since rectangles are so common, there is a special syntax for them: To add a rectangle to the current path, use the `rectangle` path construction operation. This operation should be followed by another coordinate and will append a rectangle to the path such that the previous coordinate and the next coordinates are corners of the rectangle. So, let us add two rectangles to the picture:

接下来我们想要的是背景中的网格。有几种方法可以实现。例如，可以绘制许多矩形。由于矩形非常常见，因此有一种特殊的语法可以用于绘制矩形：要将矩形添加到当前路径中，请使用`rectangle` 路径构造操作。此操作后应跟另一个坐标，并将矩形附加到路径中，以使前一个坐标和下一个坐标成为矩形的角。让我们在图片中添加两个矩形：



```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (0,0) rectangle (0.5,0.5);
  \draw (-0.5,-0.5) rectangle (-1,-1);
\end{tikzpicture}
```

While this may be nice in other situations, this is not really leading anywhere with Karl's problem: First, we would need an awful lot of these rectangles and then there is the border that is not "closed".

虽然在其他情况下这可能很好，但对于 Karl 的问题来说，这并没有真正解决问题：首先，我们需要大量这些矩形，其次边界并没有被 "闭合"。

So, Karl is about to resort to simply drawing four vertical and four horizontal lines using the nice `\draw` command, when he learns that there is a `grid` path construction operation.

所以，当 Karl 打算使用简单的`\draw` 命令绘制四条垂直线和四条水平线时，他得知有一个`grid` 路径构造操作。

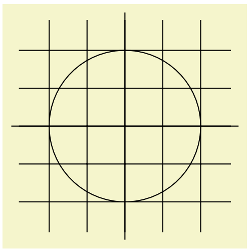
2.7 Grid Path Construction
网格路径构造

The `grid` path operation adds a grid to the current path. It will add lines making up a grid that fills the rectangle whose one corner is the current point and whose other corner is the point following the `grid` operation. For example, the code `\tikz \draw[step=2pt] (0,0) grid (10pt,10pt);` produces . Note how the optional argument for `\draw` can be used to specify a grid width (there are also `xstep` and `ystep` to define the steppings independently). As Karl will learn soon, there are *lots* of things that can be influenced using such options.

`grid` 路径操作将网格添加到当前路径中。它将添加构成填充由当前点和`grid` 操作后的点确定的矩形的网格线。例如，代码`\tikz \draw[step=2pt] (0,0) grid (10pt,10pt);` 将产生 。请注意，`\draw` 的可选参数可用于指定网格宽度（也有`xstep` 和`ystep` 可单独定义步进）。正如 Karl 很快会了解到的，有很多事情可以通过这些选项来控制。

For Karl, the following code could be used:

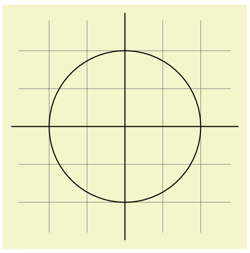
对于 Karl 来说，可以使用以下代码：



```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw[step=.5cm] (-1.4,-1.4) grid (1.4,1.4);
\end{tikzpicture}
```

Having another look at the desired picture, Karl notices that it would be nice for the grid to be more subdued. (His son told him that grids tend to be distracting if they are not subdued.) To subdue the grid, Karl adds two more options to the `\draw` command that draws the grid. First, he uses the color `gray` for the grid lines. Second, he reduces the line width to `very thin`. Finally, he swaps the ordering of the commands so that the grid is drawn first and everything else on top.

再次查看所需的图片，Karl 注意到网格更加柔和会更好（他的儿子告诉他，如果网格没有柔和处理，会分散注意力）。为了使网格柔和，Karl 向绘制网格的`\draw` 命令添加了两个选项。首先，他使用`gray` 颜色绘制网格线。其次，他将线宽减小为`very thin`。最后，他交换了命令的顺序，使网格先绘制，其他内容在其上绘制。



```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

2.8 Adding a Touch of Style
添加一点样式

Instead of the options `gray,very thin` Karl could also have said `help lines`. *Styles* are predefined sets of options that can be used to organize how a graphic is drawn. By saying `help lines` you say "use the style that I (or someone else) has set for drawing help lines". If Karl decides, at some later point, that grids should be drawn, say, using the color `blue!50` instead of `gray`, he could provide the following option somewhere:

卡尔可以选择使用选项`gray,very thin`，也可以选择`help lines`。*样式*是预定义的选项集，用于组织绘制图形的方式。通过使用`help lines`，你在实际上是在说"使用我（或其他人）为绘制辅助线设置的样式"。如果卡尔在以后的某个时间决定，网格应该使用颜色`blue!50` 而不是`gray` 来绘制，他可以在某个地方提供以下选项：

```
help lines/.style={color=blue!50,very thin}
```

The effect of this "style setter" is that in the current scope or environment the `help lines` option has the same effect as `color=blue!50,very thin`.

这个“样式设置器”的效果是，在当前的作用域或环境中，`help lines` 选项具有与`color=blue!50,very thin` 相同的效果。

Using styles makes your graphics code more flexible. You can change the way things look easily in a consistent manner. Normally, styles are defined at the beginning of a picture. However, you may sometimes wish to define a style globally, so that all pictures of your document can use this style. Then you can easily change the way all graphics look by changing this one style. In this situation you can use the `\tikzset` command at the beginning of the document as in

使用样式可以使你的图形代码更加灵活。你可以轻松以一致的方式改变事物的外观。通常，样式在图片的开头定义。然而，有时你可能希望全局定义一个样式，这样文档中的所有图片都可以使用这个样式。然后，你可以在文档开头使用`\tikzset` 命令，如下所示：

```
\tikzset{help lines/.style=very thin}
```

To build a hierarchy of styles you can have one style use another. So in order to define a style `Karl’s grid` that is based on the `grid` style Karl could say

为了构建样式的层次结构，你可以让一个样式使用另一个样式。因此，为了定义一个基于样式`grid` 的样式`Karl’s grid`，卡尔可以这样说：

```
\tikzset{Karl’s grid/.style={help lines,color=blue!50}}
...
\draw[Karl’s grid] (0,0) grid (5,5);
```

Styles are made even more powerful by parametrization. This means that, like other options, styles can also be used with a parameter. For instance, Karl could parameterize his grid so that, by default, it is blue, but he could also use another color.

通过参数化，样式的功能更加强大。这意味着，像其他选项一样，样式也可以带有参数。例如，卡尔可以使他的网格参数化，以便默认情况下为蓝色，但他也可以使用其他颜色。

```
\begin{tikzpicture}
  [Karl’s grid/.style  ={help lines,color=#1!50},
   Karl’s grid/.default=blue]

  \draw[Karl’s grid]     (0,0) grid (1.5,2);
  \draw[Karl’s grid=red] (2,0) grid (3.5,2);
\end{tikzpicture}
```

In this example, the definition of the style `Karl’s grid` is given as an optional argument to the `{tikzpicture}` environment. Additional styles for other elements would follow after a comma. With many styles in effect, the optional argument of the environment may easily happen to be longer than the actual contents.

在这个例子中，样式`Karl’s grid` 的定义作为可选参数给出，放在`{tikzpicture}` 环境中。其他元素的附加样式将在逗号后面跟随。当许多样式生效时，环境的可选参数可能比实际内容还要长。

2.9 Drawing Options 绘制选项

Karl wonders what other options there are that influence how a path is drawn. He saw already that the `color=`⟨*color*⟩ option can be used to set the line’s color. The option `draw=`⟨*color*⟩ does nearly the same, only it sets the color for the lines only and a different color can be used for filling (Karl will need this when he fills the arc for the angle).

卡尔想知道还有哪些选项会影响路径的绘制方式。他已经看到了`color=`⟨*color*⟩ 选项可以用于设置线条的颜色。选项`draw=`⟨*color*⟩ 几乎相同，只是它只设置线条的颜色，可以使用不同的颜色进行填充（当卡尔填充角度的弧时会用到这个）。

He saw that the style `very thin` yields very thin lines. Karl is not really surprised by this and neither is he surprised to learn that `thin` yields thin lines, `thick` yields thick lines, `very thick` yields very thick lines, `ultra thick` yields really, really thick lines and `ultra thin` yields lines that are so thin that low-resolution printers and displays will have trouble showing them. He wonders what gives lines of “normal” thickness. It turns out that `thin` is the correct choice, since it gives the same thickness as TEX’s `\hrule` command. Nevertheless, Karl would like to know whether there is anything “in the middle” between `thin` and `thick`. There is: `semithick`.

他已经看到 `very thin` 样式可以绘制非常细的线条。卡尔对此并不感到惊讶，他也不惊讶地了解到 `thin` 样式可以绘制细线条，`thick` 样式可以绘制粗线条，`very thick` 样式可以绘制非常粗的线条，`ultra thick` 样式可以绘制非常非常粗的线条，而 `ultra thin` 样式可以绘制非常细的线条，低分辨率的打印机和显示器

可能无法显示它们。他想知道是否有一种线条厚度介于 thin 和 thick 之间的选项。答案是有的，这就是 semithick。

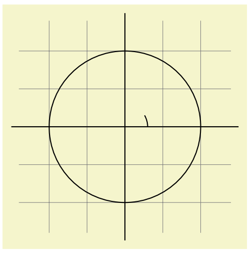
Another useful thing one can do with lines is to dash or dot them. For this, the two styles dashed and dotted can be used, yielding - - - - and ········. Both options also exist in a loose and a dense version, called loosely dashed, densely dashed, loosely dotted, and densely dotted. If he really, really needs to, Karl can also define much more complex dashing patterns with the dash pattern option, but his son insists that dashing is to be used with utmost care and mostly distracts. Karl's son claims that complicated dashing patterns are evil. Karl's students do not care about dashing patterns.

对于线条，还有一个有用的功能是虚线和点线。可以使用两个样式 dashed 和 dotted，分别绘制虚线和点线，得到的效果分别如下：- - - - 和 ········。这两个选项还有一种稀疏和稠密的版本，分别称为 loosely dashed、densely dashed、loosely dotted 和 densely dotted。如果卡尔确实需要的话，还可以使用 dash pattern 选项定义更复杂的虚线样式，但他的儿子坚持认为虚线应该小心使用，因为它们会分散注意力。卡尔的儿子声称复杂的虚线样式是有害的。卡尔的学生们对虚线样式不太关心。

2.10 Arc Path Construction 弧路径构建

Our next obstacle is to draw the arc for the angle. For this, the arc path construction operation is useful, which draws part of a circle or ellipse. This arc operation is followed by options in brackets that specify the arc. An example would be arc[start angle=10, end angle=80, radius=10pt], which means exactly what it says. Karl obviously needs an arc from 0° to 30°. The radius should be something relatively small, perhaps around one third of the circle's radius. When one uses the arc path construction operation, the specified arc will be added with its starting point at the current position. So, we first have to "get there".

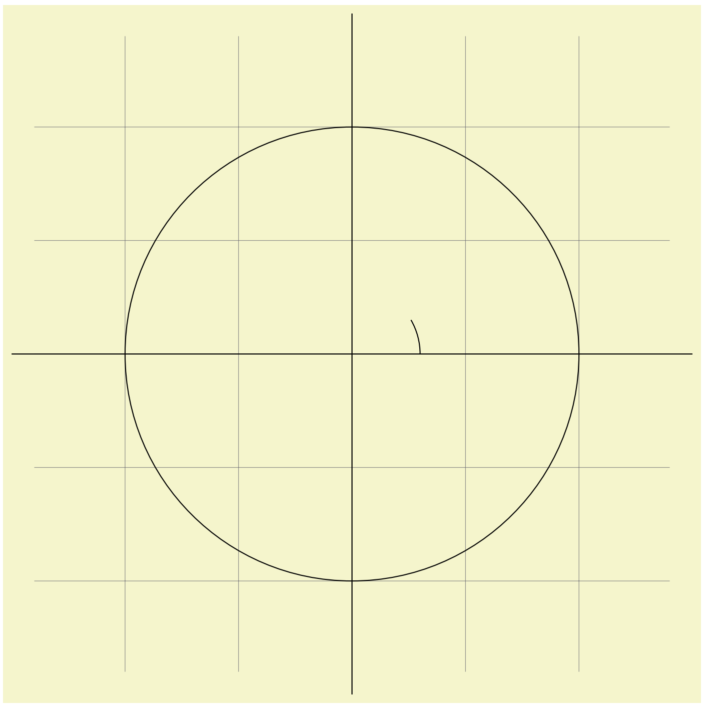
下一个障碍是绘制角度的弧。为此，arc 路径构建操作非常有用，它绘制了圆或椭圆的一部分。这个 arc 操作后面跟着方括号中指定弧的选项。一个例子是 arc[start angle=10, end angle=80, radius=10pt]，它的含义如字面意思所述。卡尔显然需要从 0° 到 30° 的弧。半径应该相对较小，可能是整个圆半径的三分之一左右。当使用弧路径构建操作时，指定的弧将从当前位置开始添加。因此，我们首先要"到达那里"。



```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

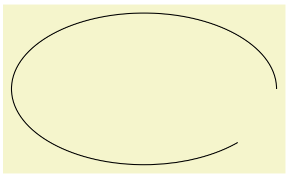
Karl thinks this is really a bit small and he cannot continue unless he learns how to do scaling. For this, he can add the [scale=3] option. He could add this option to each \draw command, but that would be awkward. Instead, he adds it to the whole environment, which causes this option to apply to everything within.

卡尔认为这个弧太小了，除非他学会如何进行缩放，否则无法继续。为此，他可以添加 [scale=3] 选项。他可以将此选项添加到每个\draw 命令，但那样会很麻烦。相反，他将其添加到整个环境中，这使得此选项适用于其中的所有内容。



```
\begin{tikzpicture}[scale=3]
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

As for circles, you can specify “two” radii in order to get an elliptical arc.
对于圆，您可以指定“两个”半径以得到椭圆弧。

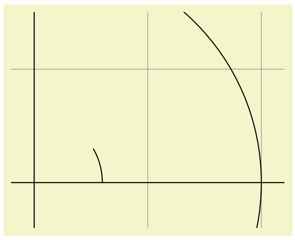


```
\tikz \draw (0,0)
  arc [start angle=0, end angle=315,
       x radius=1.75cm, y radius=1cm];
```

2.11 Clipping a Path 裁剪路径

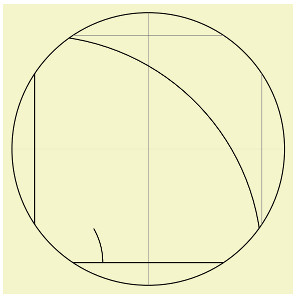
In order to save space in this manual, it would be nice to clip Karl’s graphics a bit so that we can focus on the “interesting” parts. Clipping is pretty easy in Ti*k*Z. You can use the `\clip` command to clip all subsequent drawing. It works like `\draw`, only it does not draw anything, but uses the given path to clip everything subsequently.

为了在本手册中节省空间，我们希望将卡尔的图形裁剪一下，以便我们可以专注于“有趣”的部分。在 Ti*k*Z 中，裁剪非常简单。您可以使用`\clip` 命令来裁剪所有后续的绘图。它的使用方式类似于`\draw`，只是它不绘制任何东西，而是使用给定的路径来裁剪后续的一切。



```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

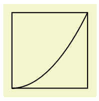
You can also do both at the same time: Draw *and* clip a path. For this, use the `\draw` command and add the `clip` option. (This is not the whole picture: You can also use the `\clip` command and add the `draw` option. Well, that is also not the whole picture: In reality, `\draw` is just a shorthand for `\path[draw]` and `\clip` is a shorthand for `\path[clip]` and you could also say `\path[draw,clip]`.) Here is an example:
您还可以同时绘制路径和剪裁路径。为此，请使用 `\draw` 命令并添加 `clip` 选项。（这还不是全部内容：您还可以使用 `\clip` 命令并添加 `draw` 选项。嗯，这还不是全部内容：实际上，`\draw` 只是 `\path[draw]` 的简写，`\clip` 是 `\path[clip]` 的简写，您也可以使用 `\path[draw,clip]`。）以下是一个示例：



```
\begin{tikzpicture}[scale=3]
  \clip[draw] (0.5,0.5) circle (.6cm);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

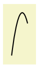
2.12 Parabola and Sine Path Construction 抛物线和正弦路径构造

Although Karl does not need them for his picture, he is pleased to learn that there are `parabola` and `sin` and `cos` path operations for adding parabolas and sine and cosine curves to the current path. For the `parabola` operation, the current point will lie on the parabola as well as the point given after the parabola operation. Consider the following example:
虽然 Karl 并不需要它们来绘制他的图片，但他很高兴地了解到存在 `parabola`、`sin` 和 `cos` 路径操作，用于在当前路径中添加抛物线和正弦曲线以及余弦曲线。对于 `parabola` 操作，当前点将位于抛物线上，以及在抛物线操作之后给出的点上。考虑以下示例：



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

It is also possible to place the bend somewhere else:
也可以将曲线弯曲到其他位置：

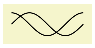


```
\tikz \draw[x=1pt,y=1pt] (0,0) parabola bend (4,16) (6,12);
```

The operations `sin` and `cos` add a sine or cosine curve in the interval $[0, \pi/2]$ such that the previous current point is at the start of the curve and the curve ends at the given end point. Here are two examples:
`sin` 和 `cos` 操作在区间 $[0, \pi/2]$ 中添加正弦曲线或余弦曲线，使得前一个当前点位于曲线的起点，曲线在给定的终点结束。以下是两个示例：

A sine curve.

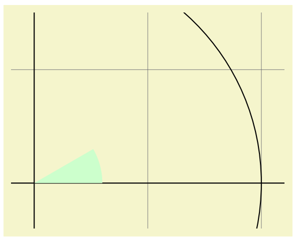
```
A sine \tikz \draw[x=1ex,y=1ex] (0,0) sin (1.57,1); curve.
```



```
\tikz \draw[x=1.57ex,y=1ex] (0,0) sin (1,1) cos (2,0) sin (3,-1) cos (4,0)
                             (0,1) cos (1,0) sin (2,-1) cos (3,0) sin (4,1);
```

2.13 Filling and Drawing 填充和绘制

Returning to the picture, Karl now wants the angle to be “filled” with a very light green. For this he uses `\fill` instead of `\draw`. Here is what Karl does:
回到图形，Karl 现在想要“填充”角度，使用非常浅的绿色。为此，他使用`\fill` 而不是`\draw`。以下是 Karl 的操作：



```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \fill[green!20!white] (0,0) -- (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm] -- (0,0);
\end{tikzpicture}
```

The color `green!20!white` means 20% green and 80% white mixed together. Such color expression are possible since Ti*k*Z uses Uwe Kern's `xcolor` package, see the documentation of that package for details on color expressions.

颜色`green!20!white` 表示 20% 的绿色和 80% 的白色混合在一起。由于 Ti*k*Z 使用了 Uwe Kern 的`xcolor` 宏包，因此可以使用这种颜色表达式，请参阅该宏包的文档以了解有关颜色表达式的详细信息。

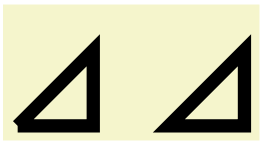
What would have happened, if Karl had not "closed" the path using `--(0,0)` at the end? In this case, the path is closed automatically, so this could have been omitted. Indeed, it would even have been better to write the following, instead:

如果 Karl 在最后没有使用`--(0,0)` "闭合" 路径会发生什么？在这种情况下，路径会自动闭合，因此可以省略这一部分。事实上，最好写成以下形式：

```
\fill[green!20!white] (0,0) -- (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm] -- cycle;
```

The `--cycle` causes the current path to be closed (actually the current part of the current path) by smoothly joining the first and last point. To appreciate the difference, consider the following example:

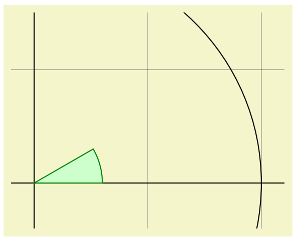
`--cycle` 通过平滑连接第一个和最后一个点来关闭当前路径（实际上是当前路径的当前部分）。为了体会到差异，请考虑以下示例：



```
\begin{tikzpicture}[line width=5pt]
  \draw (0,0) -- (1,0) -- (1,1) -- (0,0);
  \draw (2,0) -- (3,0) -- (3,1) -- cycle;
  \useasboundingbox (0,1.5); % make bounding box higher
\end{tikzpicture}
```

You can also fill and draw a path at the same time using the `\filldraw` command. This will first draw the path, then fill it. This may not seem too useful, but you can specify different colors to be used for filling and for stroking. These are specified as optional arguments like this:

你还可以同时填充和绘制路径，使用`\filldraw` 命令。这将首先绘制路径，然后填充它。这可能看起来并不太有用，但是你可以指定用于填充和描边的不同颜色。这些颜色可以作为可选参数指定，如下所示：

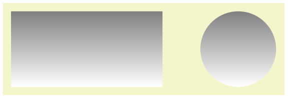


```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \filldraw[fill=green!20!white, draw=green!50!black] (0,0) -- (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\end{tikzpicture}
```

2.14 Shading 渐变

Karl briefly considers the possibility of making the angle "more fancy" by *shading* it. Instead of filling the area with a uniform color, a smooth transition between different colors is used. For this, `\shade` and `\shadedraw`, for shading and drawing at the same time, can be used:

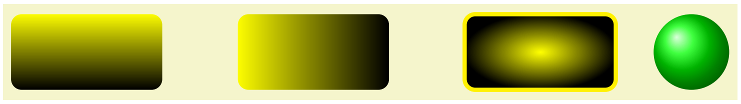
Karl 简要考虑了通过*渐变*使角度 "更加花哨" 的可能性。与使用均匀颜色填充区域不同，使用不同颜色之间的平滑过渡。为此，可以使用`\shade` 和`\shadedraw` 进行渐变和同时绘制：



```
\tikz \shade (0,0) rectangle (2,1)  (3,0.5) circle (.5cm);
```

The default shading is a smooth transition from gray to white. To specify different colors, you can use options:

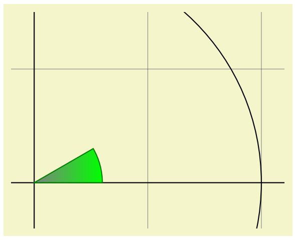
默认的渐变是从灰色到白色的平滑过渡。要指定不同的颜色，可以使用选项：



```
\begin{tikzpicture}[rounded corners,ultra thick]
  \shade[top color=yellow,bottom color=black] (0,0) rectangle +(2,1);
  \shade[left color=yellow,right color=black] (3,0) rectangle +(2,1);
  \shadedraw[inner color=yellow,outer color=black,draw=yellow] (6,0) rectangle +(2,1);
  \shade[ball color=green] (9,.5) circle (.5cm);
\end{tikzpicture}
```

For Karl, the following might be appropriate:

对于 Karl 来说，可能适合的是：



```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \shadedraw[left color=gray,right color=green, draw=green!50!black]
    (0,0) -- (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\end{tikzpicture}
```

However, he wisely decides that shadings usually only distract without adding anything to the picture.

然而，他明智地决定，渐变通常只会分散注意力，而不会为图形增添任何东西。

2.15 Specifying Coordinates 指定坐标

Karl now wants to add the sine and cosine lines. He knows already that he can use the `color=` option to set the lines' colors. So, what is the best way to specify the coordinates?

现在 Karl 想要添加正弦和余弦线。他已经知道可以使用`color=` 选项来设置线条的颜色。那么，最好的方法是如何指定坐标呢？

There are different ways of specifying coordinates. The easiest way is to say something like `(10pt,2cm)`. This means 10pt in x-direction and 2cm in y-directions. Alternatively, you can also leave out the units as in `(1,2)`, which means "one times the current x-vector plus twice the current y-vector". These vectors default to 1cm in the x-direction and 1cm in the y-direction, respectively.

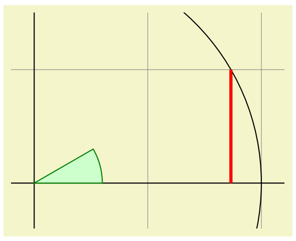
有不同的指定坐标的方法。最简单的方法是使用类似于`(10pt,2cm)` 的表示方式。这表示在 x 方向上为 10pt，在 y 方向上为 2cm。另外，你也可以省略单位，例如`(1,2)`，它表示 "当前 x 向量的一倍加上当前 y 向量的两倍"。这些向量默认分别为 1cm 和 1cm。

In order to specify points in polar coordinates, use the notation `(30:1cm)`, which means 1cm in direction 30 degree. This is obviously quite useful to "get to the point $(\cos 30°, \sin 30°)$ on the circle".

为了指定极坐标中的点，可以使用`(30:1cm)` 的表示方式，它表示方向为 30 度的 1cm。这显然非常有用，可以 "到达圆上的点 $(\cos 30°, \sin 30°)$"。

You can add a single `+` sign in front of a coordinate or two of them as in `+(0cm,1cm)` or `++(2cm,0cm)`. Such coordinates are interpreted differently: The first form means "1cm upwards from the previous specified position" and the second means "2cm to the right of the previous specified position, making this the new specified position". For example, we can draw the sine line as follows:

你可以在坐标前面加一个单独的`+` 符号，或者两个`+` 符号，例如`+(0cm,1cm)` 或`++(2cm,0cm)`。这些坐标的解释方式不同：第一种形式表示相对于之前指定的位置向上移动 1cm"，而第二种表示相对于之前指定的位置向右移动 2cm，并将其作为新指定的位置"。例如，我们可以如下绘制正弦线：



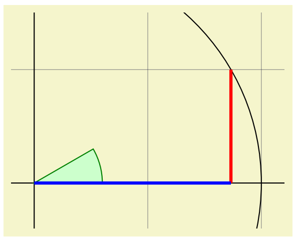
```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
      arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[red,very thick] (30:1cm) -- +(0,-0.5);
\end{tikzpicture}
```

Karl used the fact $\sin 30° = 1/2$. However, he very much doubts that his students know this, so it would be nice to have a way of specifying "the point straight down from `(30:1cm)` that lies on the x-axis". This is, indeed, possible using a special syntax: Karl can write `(30:1cm |- 0,0)`. In general, the meaning of `(`$\langle p\rangle$ `|-` $\langle q\rangle$`)` is "the intersection of a vertical line through p and a horizontal line through q".

Karl 事实上使用了 $\sin 30° = 1/2$。然而，他非常怀疑他的学生们是否知道这一点，所以最好有一种方法来指定从`(30:1cm)` 指向 x 轴下方的点"。事实上，使用一种特殊的语法是可以实现的：Karl 可以写成`(30:1cm |- 0,0)`。一般来说，`(`$\langle p\rangle$ `|-` $\langle q\rangle$`)` 的意思是通过 p 的垂直线与 q 的水平线的交点"。

Next, let us draw the cosine line. One way would be to say `(30:1cm |- 0,0) -- (0,0)`. Another way is the following: we "continue" from where the sine ends:

接下来，让我们绘制余弦线。一种方式是`(30:1cm |- 0,0) -- (0,0)`。另一种方式是：我们"延续"正弦线的结束位置：



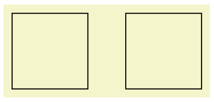
```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
      arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[red,very thick]  (30:1cm) -- +(0,-0.5);
  \draw[blue,very thick] (30:1cm) ++(0,-0.5) -- (0,0);
\end{tikzpicture}
```

Note that there is no `--` between `(30:1cm)` and `++(0,-0.5)`. In detail, this path is interpreted as follows: "First, the `(30:1cm)` tells me to move my pen to $(\cos 30°, 1/2)$. Next, there comes another coordinate specification, so I move my pen there without drawing anything. This new point is half a unit down from the last position, thus it is at $(\cos 30°, 0)$. Finally, I move the pen to the origin, but this time drawing something (because of the `--`)."

请注意，在 `(30:1cm)` 和 `++(0,-0.5)` 之间没有 `--`。具体来说，该路径的解释如下：" 首先，`(30:1cm)` 告诉我将笔移动到 $(\cos 30°, 1/2)$。接下来，出现了另一个坐标指定，所以我将笔移动到那个位置，但不画任何东西。这个新点相对于上一个位置向下移动了半个单位，因此它位于 $(\cos 30°, 0)$。最后，我将笔移动到原点，但这次画了一些东西（因为有 `--`）"。

To appreciate the difference between `+` and `++` consider the following example:

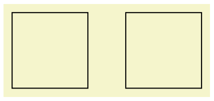
为了理解 `+` 和 `++` 之间的区别，请考虑以下示例：



```
\begin{tikzpicture}
  \def\rectanglepath{-- ++(1cm,0cm)  -- ++(0cm,1cm)  -- ++(-1cm,0cm) -- cycle}
  \draw (0,0) \rectanglepath;
  \draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

By comparison, when using a single `+`, the coordinates are different:

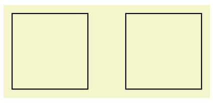
相比之下，当使用单个 `+` 时，坐标是不同的：



```
\begin{tikzpicture}
  \def\rectanglepath{-- +(1cm,0cm)  -- +(1cm,1cm)  -- +(0cm,1cm) -- cycle}
  \draw (0,0) \rectanglepath;
  \draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

Naturally, all of this could have been written more clearly and more economically like this (either with a single or a double `+`):

当然，所有这些都可以更清晰、更简洁地写成这样（无论是使用单个还是双个 `+`）：



```
\tikz \draw (0,0) rectangle +(1,1)  (1.5,0) rectangle +(1,1);
```

2.16 Intersecting Paths 相交路径

Karl is left with the line for $\tan\alpha$, which seems difficult to specify using transformations and polar coordinates. The first – and easiest – thing he can do is so simply use the coordinate `(1,{tan(30)})` since Ti*k*Z's math engine knows how to compute things like `tan(30)`. Note the added braces since, otherwise, Ti*k*Z's parser would think that the first closing parenthesis ends the coordinate (in general, you need to add braces around components of coordinates when these components contain parentheses).

Karl 手头有一个表示 $\tan\alpha$ 的直线，使用变换和极坐标来指定它似乎比较困难。他可以简单地使用坐标`(1,{tan(30)})`，因为 Ti*k*Z 的数学引擎知道如何计算类似`tan(30)` 的表达式。请注意，要加上花括号，否则 Ti*k*Z 的解析器会认为第一个闭括号结束了坐标（通常情况下，当坐标的组成部分包含括号时，需要在其周围加上花括号）。

Karl can, however, also use a more elaborate, but also more "geometric" way of computing the length of the orange line: He can specify intersections of paths as coordinates. The line for $\tan\alpha$ starts at $(1,0)$ and goes upward to a point that is at the intersection of a line going "up" and a line going from the origin through `(30:1cm)`. Such computations are made available by the `intersections` library.

然而，Karl 也可以使用一种更复杂、更"几何"的方式来计算橙色线段的长度：他可以将路径的交点指定为坐标。表示 $\tan\alpha$ 的线段从 $(1,0)$ 开始向上延伸，直到一个点，该点是向"上"延伸的线和经过原点的经过`(30:1cm)` 的线的交点。这样的计算可以通过`intersections` 库来实现。

What Karl must do is to create two "invisible" paths that intersect at the position of interest. Creating paths that are not otherwise seen can be done using the `\path` command without any options like `draw` or `fill`. Then, Karl can add the `name path` option to the path for later reference. Once the paths have been constructed, Karl can use the `name intersections` to assign names to the coordinate for later reference.

Karl 需要做的是创建两条在感兴趣位置相交的"不可见"路径。使用不带任何选项（如`draw` 或`fill`）的`\path` 命令可以创建不可见的路径。然后，Karl 可以为路径添加`name path` 选项以供以后引用。一旦构建完路径，Karl 可以使用`name intersections` 为坐标赋予名称以供以后引用。

```
\path [name path=upward line] (1,0) -- (1,1);
\path [name path=sloped line] (0,0) -- (30:1.5cm); % a bit longer, so that there is an intersection

% (add `\usetikzlibrary{intersections}' after loading tikz in the preamble)
\draw [name intersections={of=upward line and sloped line, by=x}]
  [very thick,orange] (1,0) -- (x);
```

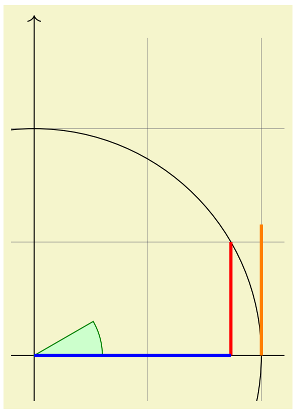
2.17 Adding Arrow Tips 添加箭头标记

Karl now wants to add the little arrow tips at the end of the axes. He has noticed that in many plots, even in scientific journals, these arrow tips seem to be missing, presumably because the generating programs cannot produce them. Karl thinks arrow tips belong at the end of axes. His son agrees. His students do not care about arrow tips.

Karl 现在想在坐标轴的末端添加一些小箭头标记。他注意到在许多绘图中，即使在科学期刊中，这些箭头标记似乎也缺失了，可能是因为生成程序无法产生它们。Karl 认为箭头标记应该放在坐标轴的末端。他的儿子也同意。他的学生们则对箭头标记不感兴趣。

It turns out that adding arrow tips is pretty easy: Karl adds the option `->` to the drawing commands for the axes:

添加箭头标记其实非常简单：Karl 只需在绘制坐标轴的命令中添加选项`->`：



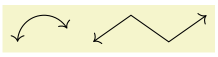
```
\usetikzlibrary {intersections}
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,1.51);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw[->] (-1.5,0) -- (1.5,0);
  \draw[->] (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
       arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[red,very thick]    (30:1cm) -- +(0,-0.5);
  \draw[blue,very thick]   (30:1cm) ++(0,-0.5) -- (0,0);

  \path [name path=upward line] (1,0) -- (1,1);
  \path [name path=sloped line] (0,0) -- (30:1.5cm);
  \draw [name intersections={of=upward line and sloped line, by=x}]
        [very thick,orange] (1,0) -- (x);
\end{tikzpicture}
```

If Karl had used the option `<-` instead of `->`, arrow tips would have been put at the beginning of the path. The option `<->` puts arrow tips at both ends of the path.

如果卡尔使用选项 `<-` 而不是 `->`，箭头将会放在路径的起始位置。选项 `<->` 在路径的两端放置箭头。

There are certain restrictions to the kind of paths to which arrow tips can be added. As a rule of thumb, you can add arrow tips only to a single open "line". For example, you cannot add tips to, say, a rectangle or a circle. However, you can add arrow tips to curved paths and to paths that have several segments, as in the following examples:

对于可以添加箭头的路径类型有一些限制。一般来说，你只能在一个单独的"线段"上添加箭头。例如，你不能在矩形或圆形上添加箭头。但是，你可以在曲线路径和具有多个线段的路径上添加箭头，如下面的示例所示：



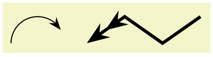
```
\begin{tikzpicture}
  \draw [<->] (0,0) arc [start angle=180, end angle=30, radius=10pt];
  \draw [<->] (1,0) -- (1.5cm,10pt) -- (2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```

Karl has a more detailed look at the arrow that Ti*k*Z puts at the end. It looks like this when he zooms it: $\rightarrow$. The shape seems vaguely familiar and, indeed, this is exactly the end of TEX's standard arrow used in something like $f\colon A \to B$.

卡尔仔细观察了 Ti*k*Z 放在路径末端的箭头。当他放大观察时，它看起来是这样的：$\rightarrow$。这个形状似乎有点熟悉，实际上，这正是类似于 $f\colon A \to B$ 中 TEX 标准箭头的末端。

Karl likes the arrow, especially since it is not "as thick" as the arrows offered by many other packages. However, he expects that, sometimes, he might need to use some other kinds of arrow. To do so, Karl can say `>=`⟨*kind of end arrow tip*⟩, where ⟨*kind of end arrow tip*⟩ is a special arrow tip specification. For example, if Karl says `>=Stealth`, then he tells Ti*k*Z that he would like "stealth-fighter-like" arrow tips:

卡尔喜欢这个箭头，特别是它不像其他许多包提供的箭头那么"粗"。然而，他预计有时可能需要使用其他类型的箭头。为了这样做，卡尔可以使用 `>=`⟨ *箭头类型*⟩，其中 ⟨ *箭头类型*⟩ 是一种特殊的箭头类型说明。例如，如果卡尔使用 `>=Stealth`，则表示他想要"隐形战斗机"样式的箭头：



```
\usetikzlibrary {arrows.meta}
\begin{tikzpicture}[>=Stealth]
  \draw [->] (0,0) arc [start angle=180, end angle=30, radius=10pt];
  \draw [<<-,very thick] (1,0) -- (1.5cm,10pt) -- (2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```

Karl wonders whether such a military name for the arrow type is really necessary. He is not really mollified when his son tells him that Microsoft's PowerPoint uses the same name. He decides to have his students discuss this at some point.

卡尔想知道是否真的有必要给箭头类型取一个军事名称。当他的儿子告诉他微软的 PowerPoint 也使用相同的名称时，他并没有完全安心。他决定在某个时候让学生们讨论这个问题。

In addition to `Stealth` there are several other predefined kinds of arrow tips Karl can choose from, see Section ??. Furthermore, he can define arrows types himself, if he needs new ones.

除了 `Stealth`，卡尔还可以从几种预定义的箭头类型中进行选择，详见第 ??节。此外，如果需要，他还可以自定义箭头类型。

2.18 Scoping
作用范围

Karl saw already that there are numerous graphic options that affect how paths are rendered. Often, he would like to apply certain options to a whole set of graphic commands. For example, Karl might wish to draw three paths using a `thick` pen, but would like everything else to be drawn “normally”.

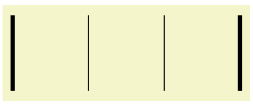
卡尔已经注意到有许多图形选项会影响路径的渲染方式。通常情况下，他希望将某些选项应用到一整组图形命令中。例如，卡尔可能希望使用粗线绘制三条路径，但希望其他所有东西都以“正常”的方式绘制。

If Karl wishes to set a certain graphic option for the whole picture, he can simply pass this option to the `\tikz` command or to the `{tikzpicture}` environment (Gerda would pass the options to `\tikzpicture` and Hans passes them to `\starttikzpicture`). However, if Karl wants to apply graphic options to a local group, he put these commands inside a `{scope}` environment (Gerda uses `\scope` and `\endscope`, Hans uses `\startscope` and `\stopscope`). This environment takes graphic options as an optional argument and these options apply to everything inside the scope, but not to anything outside.

如果卡尔希望为整个图形设置某个图形选项，他可以将该选项直接传递给 `\tikz` 命令或 `{tikzpicture}` 环境（格尔达会将选项传递给 `\tikzpicture`，汉斯会将选项传递给 `\starttikzpicture`）。然而，如果卡尔想要将图形选项应用于局部分组，他可以将这些命令放在 `{scope}` 环境内（格尔达使用 `\scope` 和 `\endscope`，汉斯使用 `\startscope` 和 `\stopscope`）。这个环境接受一个可选的图形选项参数，并且这些选项应用于范围内的所有内容，但不影响范围外的内容。

Here is an example:

以下是一个示例：



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

Scoping has another interesting effect: Any changes to the clipping area are local to the scope. Thus, if you say `\clip` somewhere inside a scope, the effect of the `\clip` command ends at the end of the scope. This is useful since there is no other way of “enlarging” the clipping area.

作用范围还有另一个有趣的效果：对裁剪区域的任何更改都仅限于该范围内。因此，如果你在某个作用范围内使用 `\clip` 命令，它的效果将在作用范围结束时结束。这是有用的，因为没有其他方法“扩大”裁剪区域。

Karl has also already seen that giving options to commands like `\draw` apply only to that command. It turns out that the situation is slightly more complex. First, options to a command like `\draw` are not really options to the command, but they are “path options” and can be given anywhere on the path. So, instead of `\draw[thin] (0,0) -- (1,0);` one can also write `\draw (0,0) [thin] -- (1,0);` or `\draw (0,0) -- (1,0) [thin];`; all of these have the same effect. This might seem strange since in the last case, it would appear that the `thin` should take effect only “after” the line from $(0,0)$ to $(1,0)$ has been drawn. However, most graphic options only apply to the whole path. Indeed, if you say both `thin` and `thick` on the same path, the last option given will “win”.

卡尔已经注意到，给类似 `\draw` 的命令添加选项只对该命令起作用。事实证明，情况稍微复杂一些。首先，给 `\draw` 这样的命令添加的选项实际上不是该命令的选项，而是“路径选项”，可以在路径的任何位置给出。因此，可以用 `\draw[thin] (0,0) -- (1,0);`，也可以写作 `\draw (0,0) [thin] -- (1,0);` 或 `\draw (0,0) -- (1,0) [thin];`；所有这些都具有相同的效果。这可能看起来有些奇怪，因为在最后一种情况下，似乎 `thin` 应该只在从 $(0,0)$ 到 $(1,0)$ 的线段绘制“之后”才生效。然而，大多数图形选项只适用于整个路径。实际上，如果在同一路径上同时使用 `thin` 和 `thick`，最后给出的选项将“胜出”。

When reading the above, Karl notices that only “most” graphic options apply to the whole path. Indeed, all transformation options do *not* apply to the whole path, but only to “everything following them on the path”. We will have a more detailed look at this in a moment. Nevertheless, all options given during a path construction apply only to this path.

在阅读上述内容时，卡尔注意到只有“大多数”图形选项适用于整个路径。实际上，所有的变换选项不适用于整个路径，而是只适用于“路径上紧随其后的所有内容”。我们稍后将详细研究这个问题。然而，路径构造过程中给出的所有选项都只适用于该路径。

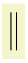
2.19 Transformations
变换

When you specify a coordinate like (1cm,1cm), where is that coordinate placed on the page? To determine the position, Ti*k*Z, TEX, and PDF or PostScript all apply certain transformations to the given coordinate in order to determine the final position on the page.

当你指定一个像 (1cm,1cm) 这样的坐标时，该坐标在页面上的哪个位置？为了确定位置，Ti*k*Z、TEX 和 PDF 或 PostScript 都会对给定的坐标应用一定的变换，以确定最终在页面上的位置。

Ti*k*Z provides numerous options that allow you to transform coordinates in Ti*k*Z's private coordinate system. For example, the xshift option allows you to shift all subsequent points by a certain amount:

Ti*k*Z 提供了许多选项，允许你在 Ti*k*Z 的私有坐标系统中进行坐标变换。例如，xshift 选项允许你将所有后续点向右平移一定量：



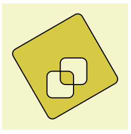
```
\tikz \draw (0,0) -- (0,0.5) [xshift=2pt] (0,0) -- (0,0.5);
```

It is important to note that you can change transformation "in the middle of a path", a feature that is not supported by PDF or PostScript. The reason is that Ti*k*Z keeps track of its own transformation matrix.

重要的是要注意，你可以在"路径中间"更改变换，而这是 PDF 或 PostScript 不支持的功能。原因是 Ti*k*Z 跟踪自己的变换矩阵。

Here is a more complicated example:

以下是一个更复杂的例子：



```
\begin{tikzpicture}[even odd rule,rounded corners=2pt,x=10pt,y=10pt]
  \filldraw[fill=yellow!80!black] (0,0)   rectangle (1,1)
        [xshift=5pt,yshift=5pt]   (0,0)   rectangle (1,1)
                    [rotate=30]   (-1,-1) rectangle (2,2);
\end{tikzpicture}
```

The most useful transformations are xshift and yshift for shifting, shift for shifting to a given point as in shift={(1,0)} or shift={+(0,0)} (the braces are necessary so that TEX does not mistake the comma for separating options), rotate for rotating by a certain angle (there is also a rotate around for rotating around a given point), scale for scaling by a certain factor, xscale and yscale for scaling only in the x- or y-direction (xscale=-1 is a flip), and xslant and yslant for slanting. If these transformation and those that I have not mentioned are not sufficient, the cm option allows you to apply an arbitrary transformation matrix. Karl's students, by the way, do not know what a transformation matrix is.

最常用的变换是 xshift 和 yshift 用于平移，shift 用于平移到给定点，如 shift={(1,0)} 或 shift={+(0,0)}（大括号是必需的，以免 TEX 将逗号误认为是选项分隔符），rotate 用于按一定角度旋转（还有一个 rotate around 用于围绕给定点旋转），scale 用于按一定比例缩放，xscale 和 yscale 用于仅在 x-方向或 y-方向缩放（xscale=-1 是翻转），xslant 和 yslant 用于倾斜。如果这些变换和我没有提到的其他变换不足够使用，cm 选项允许你应用任意的变换矩阵。顺便说一下，卡尔的学生们并不知道什么是变换矩阵。

2.20 Repeating Things: For-Loops
重复操作：For 循环

Karl's next aim is to add little ticks on the axes at positions −1, −1/2, 1/2, and 1. For this, it would be nice to use some kind of "loop", especially since he wishes to do the same thing at each of these positions. There are different packages for doing this. LATEX has its own internal command for this, pstricks comes along with the powerful \multido command. All of these can be used together with Ti*k*Z, so if you are familiar with them, feel free to use them. Ti*k*Z introduces yet another command, called \foreach, which I introduced since I could never remember the syntax of the other packages. \foreach is defined in the package pgffor and can be used independently of Ti*k*Z, but Ti*k*Z includes it automatically.

Karl 的下一个目标是在坐标轴上添加小刻度，位置分别为 −1，−1/2，1/2 和 1。为此，最好使用一种"循环"机制，特别是因为他希望在每个位置上执行相同的操作。有不同的包可以实现这一点。LATEX 内部有自己的命令，pstricks 包则提供了强大的 \multido 命令。所有这些命令都可以与 Ti*k*Z 一起使用，所以如果你熟悉它们，可以随意使用。Ti*k*Z 还引入了另一个命令，称为 \foreach，我引入它是因为我总是记不住其他包的语法。\foreach 定义在 pgffor 包中，可以独立于 Ti*k*Z 使用，但 Ti*k*Z 会自动包含它。

In its basic form, the \foreach command is easy to use:

\foreach 命令的基本用法很简单：

$x = 1$, $x = 2$, $x = 3$,

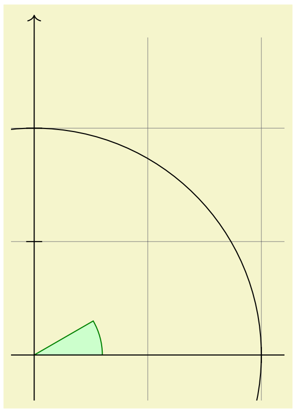
```
\foreach \x in {1,2,3} {$x =\x$, }
```

The general syntax is `\foreach` ⟨*variable*⟩ `in {`⟨*list of values*⟩`}` ⟨*commands*⟩. Inside the ⟨*commands*⟩, the ⟨*variable*⟩ will be assigned to the different values. If the ⟨*commands*⟩ do not start with a brace, everything up to the next semicolon is used as ⟨*commands*⟩.

它的一般语法是 `\foreach` ⟨ *变量*⟩ `in {`⟨ *值列表*⟩`}` ⟨ *命令*⟩。在 ⟨ *命令*⟩ 中，⟨ *变量*⟩ 将被赋予不同的值。如果 ⟨ *命令*⟩ 不以大括号开头，那么直到下一个分号之前的所有内容都将用作 ⟨ *命令*⟩。

For Karl and the ticks on the axes, he could use the following code:

对于 Karl 和坐标轴上的刻度，他可以使用以下代码：



```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,1.51);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
      arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[->] (-1.5,0) -- (1.5,0);
  \draw[->] (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];

  \foreach \x in {-1cm,-0.5cm,1cm}
    \draw (\x,-1pt) -- (\x,1pt);
  \foreach \y in {-1cm,-0.5cm,0.5cm,1cm}
    \draw (-1pt,\y) -- (1pt,\y);
\end{tikzpicture}
```

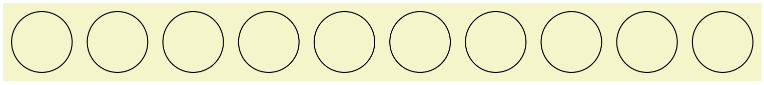
As a matter of fact, there are many different ways of creating the ticks. For example, Karl could have put the `\draw ...;` inside curly braces. He could also have used, say,

事实上，有许多不同的方法可以创建刻度。例如，Karl 可以将 `\draw ...;` 放在大括号中。他也可以使用以下方式：

```
\foreach \x in {-1,-0.5,1}
  \draw[xshift=\x cm] (0pt,-1pt) -- (0pt,1pt);
```

Karl is curious what would happen in a more complicated situation where there are, say, 20 ticks. It seems bothersome to explicitly mention all these numbers in the set for `\foreach`. Indeed, it is possible to use `...` inside the `\foreach` statement to iterate over a large number of values (which must, however, be dimensionless real numbers) as in the following example:

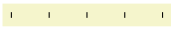
Karl 很好奇在更复杂的情况下会发生什么，比如有 20 个刻度。在 `\foreach` 中显式提及所有这些数字似乎很麻烦。的确，可以在 `\foreach` 语句中使用 `...` 来迭代一大堆值（但这些值必须是无量纲实数），如下面的例子所示：



```
\tikz \foreach \x in {1,...,10}
        \draw (\x,0) circle (0.4cm);
```

If you provide *two* numbers before the `...`, the `\foreach` statement will use their difference for the stepping:

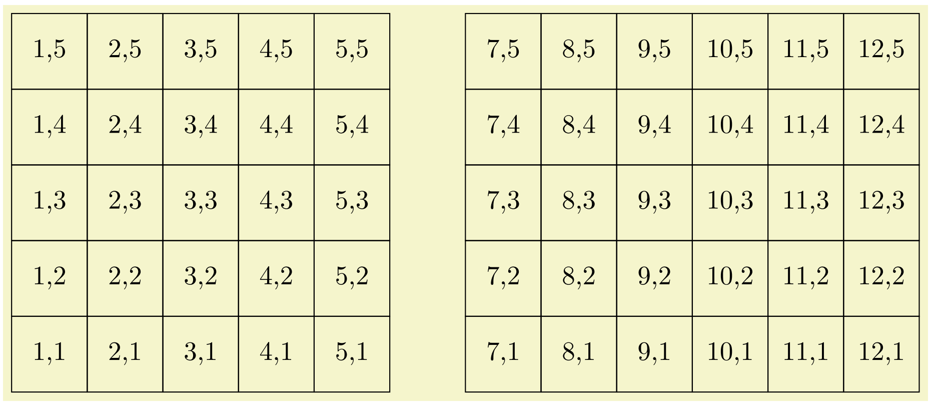
如果在 `...` 前提供*两个数字*，`\foreach` 语句将使用它们的差值作为步长：



```
\tikz \foreach \x in {-1,-0.5,...,1}
        \draw (\x cm,-1pt) -- (\x cm,1pt);
```

We can also nest loops to create interesting effects:

我们还可以嵌套循环以创建有趣的效果：



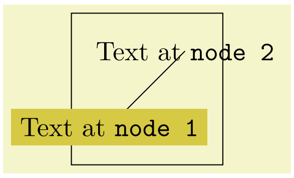
```
\begin{tikzpicture}
  \foreach \x in {1,2,...,5,7,8,...,12}
    \foreach \y in {1,...,5}
    {
      \draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
      \draw (\x,\y) node{\x,\y};
    }
\end{tikzpicture}
```

The `\foreach` statement can do even trickier stuff, but the above gives the idea.
`\foreach` 语句甚至可以执行更复杂的操作，但以上内容已经给出了基本思路。

2.21 Adding Text 添加文本

Karl is, by now, quite satisfied with the picture. However, the most important parts, namely the labels, are still missing!
卡尔对图片已经相当满意了。然而，最重要的部分，也就是标签，还是缺失的！

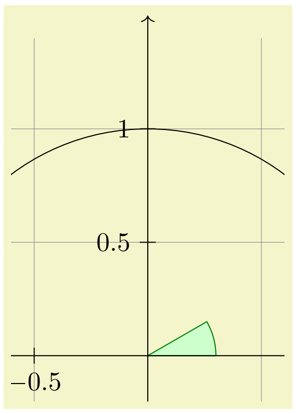
Ti*k*Z offers an easy-to-use and powerful system for adding text and, more generally, complex shapes to a picture at specific positions. The basic idea is the following: When Ti*k*Z is constructing a path and encounters the keyword `node` in the middle of a path, it reads a *node specification*. The keyword `node` is typically followed by some options and then some text between curly braces. This text is put inside a normal TEX box (if the node specification directly follows a coordinate, which is usually the case, Ti*k*Z is able to perform some magic so that it is even possible to use verbatim text inside the boxes) and then placed at the current position, that is, at the last specified position (possibly shifted a bit, according to the given options). However, all nodes are drawn only after the path has been completely drawn/filled/shaded/clipped/whatever.
Ti*k*Z 提供了一个易于使用且功能强大的系统，用于在特定位置向图片添加文本和其他复杂形状。基本思想如下：当 Ti*k*Z 构建路径并在路径中间遇到关键字 `node` 时，它会读取一个*节点规范*。关键字 `node` 通常后面跟着一些选项，然后是花括号内的一些文本。这个文本被放入一个普通的 TEX 盒子中（如果节点规范直接跟在坐标后面，这通常是情况，Ti*k*Z 能够执行一些魔法，使得甚至可以在盒子中使用抄录文本），然后放置在当前位置，即最后指定的位置（根据给定的选项可能会稍微偏移一点）。然而，所有的节点都是在路径完全绘制/填充/着色/裁剪/等等之后绘制的。



```
\begin{tikzpicture}
  \draw (0,0) rectangle (2,2);
  \draw (0.5,0.5) node [fill=yellow!80!black]
                       {Text at \verb!node 1!}
     -- (1.5,1.5) node {Text at \verb!node 2!};
\end{tikzpicture}
```

Obviously, Karl would not only like to place nodes *on* the last specified position, but also to the left or the right of these positions. For this, every node object that you put in your picture is equipped with several *anchors*. For example, the `north` anchor is in the middle at the upper end of the shape, the `south` anchor is at the bottom and the `north east` anchor is in the upper right corner. When you give the option `anchor=north`, the text will be placed such that this northern anchor will lie on the current position and the text is, thus, below the current position. Karl uses this to draw the ticks as follows:
显然，卡尔不仅希望将节点放在最后指定的位置上，还希望将其放在这些位置的左侧或右侧。为此，你在图片中放置的每个节点对象都配备了若干*锚点*。例如，`north` 锚点位于形状的上端中间，`south` 锚点位于底

部，north east 锚点位于右上角。当给定选项 anchor=north 时，文本将被放置在该北方锚点上，因此文本位于当前位置的下方。卡尔使用以下方法绘制刻度线：



```
\begin{tikzpicture}[scale=3]
  \clip (-0.6,-0.2) rectangle (0.6,1.51);
  \draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[->] (-1.5,0) -- (1.5,0);   \draw[->] (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];

  \foreach \x in {-1,-0.5,1}
    \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {$\x$};
  \foreach \y in {-1,-0.5,0.5,1}
    \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east] {$\y$};
\end{tikzpicture}
```

This is quite nice, already. Using these anchors, Karl can now add most of the other text elements. However, Karl thinks that, though "correct", it is quite counter-intuitive that in order to place something *below* a given point, he has to use the *north* anchor. For this reason, there is an option called below, which does the same as anchor=north. Similarly, above right does the same as anchor=south west. In addition, below takes an optional dimension argument. If given, the shape will additionally be shifted downwards by the given amount. So, below=1pt can be used to put a text label below some point and, additionally shift it 1pt downwards.

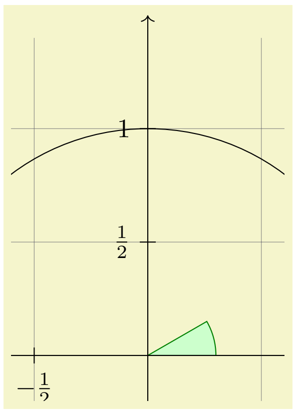
这已经相当不错了。使用这些锚点，Karl 现在可以添加大部分其他的文本元素了。然而，Karl 认为，尽管"正确"，但要在给定点的下方放置某物，他必须使用"north"锚点，这是相当不直观的。因此，有一个名为below 的选项，它与anchor=north 的作用相同。类似地，above right 与anchor=south west 的作用相同。此外，below 接受一个可选的尺寸参数。如果给定，形状将额外向下移动给定的量。因此，可以使用below=1pt 将文本标签放置在某个点的下方，并将其额外向下移动 1pt。

Karl is not quite satisfied with the ticks. He would like to have 1/2 or $\frac{1}{2}$ shown instead of 0.5, partly to show off the nice capabilities of TEX and Ti*k*Z, partly because for positions like 1/3 or π it is certainly very much preferable to have the "mathematical" tick there instead of just the "numeric" tick. His students, on the other hand, prefer 0.5 over 1/2 since they are not too fond of fractions in general.

Karl 对刻度线还不太满意。他想要显示 1/2 或 $\frac{1}{2}$，而不是 0.5，部分是为了展示 TEX 和 Ti*k*Z 的出色功能，部分是因为对于类似 1/3 或 π 的位置，使用"数学"的刻度线肯定比仅仅使用"数字"的刻度线更好。然而，他的学生们更喜欢 0.5 而不是 1/2，因为他们对分数不太感兴趣。

Karl now faces a problem: For the \foreach statement, the position \x should still be given as 0.5 since Ti*k*Z will not know where \frac{1}{2} is supposed to be. On the other hand, the typeset text should really be \frac{1}{2}. To solve this problem, \foreach offers a special syntax: Instead of having one variable \x, Karl can specify two (or even more) variables separated by a slash as in \x / \xtext. Then, the elements in the set over which \foreach iterates must also be of the form ⟨*first*⟩/⟨*second*⟩. In each iteration, \x will be set to ⟨*first*⟩ and \xtext will be set to ⟨*second*⟩. If no ⟨*second*⟩ is given, the ⟨*first*⟩ will be used again. So, here is the new code for the ticks:

现在 Karl 面临一个问题：对于\foreach 语句，位置\x 仍然应该给定为0.5，因为 Ti*k*Z 不知道\frac{1}{2} 应该在哪里。另一方面，排版的文本应该真正是\frac{1}{2}。为了解决这个问题，\foreach 提供了一种特殊的语法：Karl 可以指定两个（甚至更多）由斜杠分隔的变量，如\x / \xtext。然后，\foreach 迭代的集合中的元素必须以⟨*first*⟩/⟨*second*⟩ 的形式。在每次迭代中，\x 将被设置为⟨*first*⟩，\xtext 将被设置为⟨*second*⟩。如果没有给出⟨*second*⟩，则再次使用⟨*first*⟩。因此，这是刻度线的新代码：



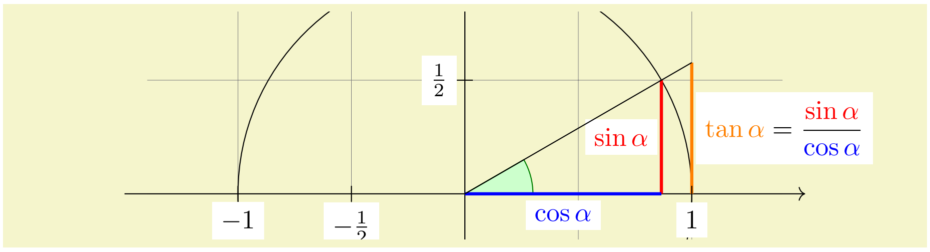
```
\begin{tikzpicture}[scale=3]
  \clip (-0.6,-0.2) rectangle (0.6,1.51);
  \draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
      arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[->] (-1.5,0) -- (1.5,0); \draw[->] (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];

  \foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
    \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {$\xtext$};
  \foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
    \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east] {$\ytext$};
\end{tikzpicture}
```

Karl is quite pleased with the result, but his son points out that this is still not perfectly satisfactory: The grid and the circle interfere with the numbers and decrease their legibility. Karl is not very concerned by this (his students do not even notice), but his son insists that there is an easy solution: Karl can add the `[fill=white]` option to fill out the background of the text shape with a white color.

Karl 对结果感到相当满意，但他的儿子指出这仍然不完全令人满意：网格和圆圈会干扰数字并降低其可读性。Karl 对此并不太担心（他的学生甚至都没注意到），但他的儿子坚持认为有一个简单的解决方案：Karl 可以在文本形状后面添加`[fill=white]` 选项，用白色填充文本形状的背景。

The next thing Karl wants to do is to add the labels like $\sin\alpha$. For this, he would like to place a label "in the middle of the line". To do so, instead of specifying the label `node {$\sin\alpha$}` directly after one of the endpoints of the line (which would place the label at that endpoint), Karl can give the label directly after the `--`, before the coordinate. By default, this places the label in the middle of the line, but the `pos=` options can be used to modify this. Also, options like `near start` and `near end` can be used to modify this position:

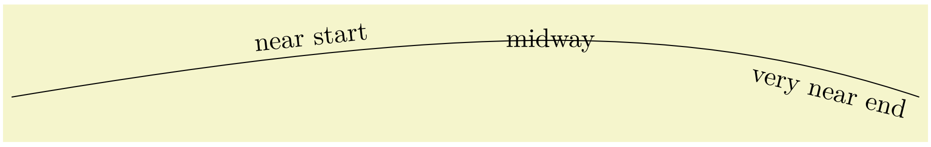
Karl 接下来想要做的是添加类似于 $\sin\alpha$ 的标签。为此，他想要将标签放在 "线的中间"。为了实现这一点，Karl 不是直接在线的一个端点之后指定标签`node {$\sin\alpha$}`（这会将标签放在该端点处），而是在`--`之后、坐标之前直接给出标签。默认情况下，这将将标签放在线的中间，但可以使用`pos=` 选项来修改这个位置。此外，还可以使用类似于`near start` 和`near end` 的选项来修改这个位置：



```
\usetikzlibrary {intersections}
\begin{tikzpicture}[scale=3]
  \clip (-2,-0.2) rectangle (2,0.8);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[->] (-1.5,0) -- (1.5,0) coordinate (x axis);
  \draw[->] (0,-1.5) -- (0,1.5) coordinate (y axis);
  \draw (0,0) circle [radius=1cm];

  \draw[very thick,red]
    (30:1cm) -- node[left=1pt,fill=white] {$\sin \alpha$} (30:1cm |- x axis);
  \draw[very thick,blue]
    (30:1cm |- x axis) -- node[below=2pt,fill=white] {$\cos \alpha$} (0,0);
  \path [name path=upward line] (1,0) -- (1,1);
  \path [name path=sloped line] (0,0) -- (30:1.5cm);
  \draw [name intersections={of=upward line and sloped line, by=t}]
    [very thick,orange] (1,0) -- node [right=1pt,fill=white]
    {$\displaystyle \tan \alpha \color{black}=
      \frac{{\color{red}\sin \alpha}}{\color{blue}\cos \alpha}$} (t);

  \draw (0,0) -- (t);

  \foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
    \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north,fill=white] {$\xtext$};
  \foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
    \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east,fill=white] {$\ytext$};
\end{tikzpicture}
```

You can also position labels on curves and, by adding the `sloped` option, have them rotated such that they match the line's slope. Here is an example:

您还可以在曲线上放置标签，并通过添加`sloped` 选项使其旋转，以与线的斜率匹配。以下是一个示例：



```
\begin{tikzpicture}
  \draw (0,0) .. controls (6,1) and (9,1) ..
    node[near start,sloped,above] {near start}
    node {midway}
    node[very near end,sloped,below] {very near end} (12,0);
\end{tikzpicture}
```

It remains to draw the explanatory text at the right of the picture. The main difficulty here lies in limiting the width of the text "label", which is quite long, so that line breaking is used. Fortunately, Karl can use the option `text width=6cm` to get the desired effect. So, here is the full code:

接下来需要在图片右侧绘制解释性文本。这里的主要困难在于限制文本 "label" 的宽度，因为它相当长，所以需要使用换行。幸运的是，卡尔可以使用选项`text width=6cm` 来实现所需的效果。因此，以下是完整的代码：

```
\begin{tikzpicture}
  [scale=3,line cap=round,
  % Styles
  axes/.style=,
  important line/.style={very thick},
  information text/.style={rounded corners,fill=red!10,inner sep=1ex}]

  % Colors
  \colorlet{anglecolor}{green!50!black}
  \colorlet{sincolor}{red}
  \colorlet{tancolor}{orange!80!black}
  \colorlet{coscolor}{blue}

  % The graphic
  \draw[help lines,step=0.5cm] (-1.4,-1.4) grid (1.4,1.4);

  \draw (0,0) circle [radius=1cm];

  \begin{scope}[axes]
    \draw[->] (-1.5,0) -- (1.5,0) node[right] {$x$} coordinate(x axis);
    \draw[->] (0,-1.5) -- (0,1.5) node[above] {$y$} coordinate(y axis);

    \foreach \x/\xtext in {-1, -.5/-\frac{1}{2}, 1}
      \draw[xshift=\x cm] (0pt,1pt) -- (0pt,-1pt) node[below,fill=white] {$\xtext$};

    \foreach \y/\ytext in {-1, -.5/-\frac{1}{2}, .5/\frac{1}{2}, 1}
      \draw[yshift=\y cm] (1pt,0pt) -- (-1pt,0pt) node[left,fill=white] {$\ytext$};
  \end{scope}

  \filldraw[fill=green!20,draw=anglecolor] (0,0) -- (3mm,0pt)
    arc [start angle=0, end angle=30, radius=3mm];
  \draw (15:2mm) node[anglecolor] {$\alpha$};

  \draw[important line,sincolor]
    (30:1cm) -- node[left=1pt,fill=white] {$\sin \alpha$} (30:1cm |- x axis);

  \draw[important line,coscolor]
    (30:1cm |- x axis) -- node[below=2pt,fill=white] {$\cos \alpha$} (0,0);

  \path [name path=upward line] (1,0) -- (1,1);
  \path [name path=sloped line] (0,0) -- (30:1.5cm);
  \draw [name intersections={of=upward line and sloped line, by=t}]
    [very thick,orange] (1,0) -- node [right=1pt,fill=white]
    {$\displaystyle \tan \alpha \color{black}=
      \frac{{\color{red}\sin \alpha}}{\color{blue}\cos \alpha}$} (t);

  \draw (0,0) -- (t);

  \draw[xshift=1.85cm]
    node[right,text width=6cm,information text]
    {
      The {\color{anglecolor} angle $\alpha$} is $30^\circ$ in the
      example ($\pi/6$ in radians). The {\color{sincolor}sine of
        $\alpha$}, which is the height of the red line, is
      \[
      {\color{sincolor} \sin \alpha} = 1/2.
      \]
      By the Theorem of Pythagoras ...
    };
\end{tikzpicture}
```

2.22 Pics: The Angle Revisited 图像：角度再探

Karl expects that the code of certain parts of the picture he created might be so useful that he might wish to reuse them in the future. A natural thing to do is to create TEX macros that store the code he wishes to reuse. However, Ti*k*Z offers another way that is integrated directly into its parser: pics!

Karl 预计他创建的图片的某些部分的代码可能非常有用，他未来可能希望重用它们。一个自然的做法是创建 TEX 宏来存储他希望重用的代码。然而，Ti*k*Z 提供了另一种直接集成到其解析器中的方法：pics（图片）!

A “pic” is “not quite a full picture”, hence the short name. The idea is that a pic is simply some code that you can add to a picture at different places using the `pic` command whose syntax is almost identical to the `node` command. The main difference is that instead of specifying some text in curly braces that should be shown, you specify the name of a predefined picture that should be shown.

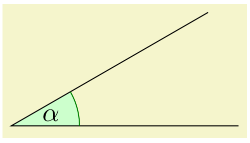
pic“（图片）是不完整的图片”，因此名字很短。这个想法是，pic 只是一些代码，你可以使用与`node` 命令几乎相同的语法，将它们添加到不同的位置的图片中。主要区别在于，你不是在花括号中指定要显示的某些文本，而是指定应该显示的预定义图片的名称。

Defining new pics is easy enough, see Section **??**, but right now we just want to use one such predefined pic: the `angle` pic. As the name suggests, it is a small drawing of an angle consisting of a little wedge and an arc together with some text (Karl needs to load the `angles` library and the `quotes` for the following examples). What makes this pic useful is the fact that the size of the wedge will be computed automatically.

定义新的 pic 非常简单，请参见第 **??**节，但现在我们只想使用一个这样的预定义 pic：`angle`（角度）pic。顾名思义，它是一个小角度的绘图，由一个小楔和一段弧线以及一些文本组成（Karl 需要加载`angles` 库和`quotes` 用于以下示例）。使得这个 pic 有用的是楔形的尺寸将自动计算。

The `angle` pic draws an angle between the two lines BA and BC, where A, B, and C are three coordinates. In our case, B is the origin, A is somewhere on the x-axis and C is somewhere on a line at 30°.

`angle`（角度）pic 在两条线 BA 和 BC 之间绘制一个角度，其中 A，B 和 C 是三个坐标。在我们的例子中，B 是原点，A 在 x 轴上的某处，C 在 30° 处的一条线上。



```
\usetikzlibrary {angles,quotes}
\begin{tikzpicture}[scale=3]
  \coordinate (A) at (1,0);
  \coordinate (B) at (0,0);
  \coordinate (C) at (30:1cm);

  \draw (A) -- (B) -- (C)
        pic [draw=green!50!black, fill=green!20, angle radius=9mm,
             "$\alpha$"] {angle = A--B--C};
\end{tikzpicture}
```

Let us see, what is happening here. First we have specified three *coordinates* using the `\coordinate` command. It allows us to name a specific coordinate in the picture. Then comes something that starts as a normal `\draw`, but then comes the `pic` command. This command gets lots of options and, in curly braces, comes the most important point: We specify that we want to add an `angle` pic and this angle should be between the points we named `A`, `B`, and `C` (we could use other names). Note that the text that we want to be shown in the pic is specified in quotes inside the options of the `pic`, not inside the curly braces.

让我们看看这里发生了什么。首先，我们使用`\coordinate` 命令指定了三个坐标。它允许我们为图片中的特定坐标命名。然后就是一个看起来像普通的`\draw` 命令，但接着是`pic` 命令。这个命令有很多选项，在花括号中最重要的是：我们指定要添加一个`angle`（角度）pic，并且这个角度应该在我们命名为`A`、`B` 和`C` 的点之间（我们可以使用其他名称）。请注意，我们希望在 pic 中显示的文本是在`pic` 的选项中用引号指定的，而不是在花括号中。

To learn more about pics, please see Section **??**.

要了解有关 pics 的更多信息，请参见第 **??**节。

```
\setcounter{section}{2}
\setcounter{subsection}{22}
\setcounter{subsubsection}{0}
```