Advanced LaTeX

Dan Parker[*] and David Schwein[†]

Spring 2015

Welcome to the second the Brown Science Center's LaTeXWorkshops! This workshop covers advanced material in LaTeX, including: advanced mathematics commands, making your own custom commands, tables and bibliography management. Additionally, these notes cover some individual packages which come in handy for various fields, including the `siunitx` package for formatting quantities with units, the `mhchem` package for displaying chemical equations, the `amsthm` package for formatting theorems, proofs, and similar mathematical conventions, and — last, but certainly not least — the incredible `TikZ` package for making graphics inside TeX.

1 Advanced Mathematics

Let's start off with talking about some more mathematics commands in LaTeX. These are classified as "advanced" not because they're more difficult, but because they're a little more obscure and used less often. You will need the following packages for the commands in this section: `amsmath, amssymb`. Load them with the command

```
\usepackage{amsmath}
\usepackage{amssymb}
```

somewhere in the preamble of your document.

1.1 Aligned Objects: Matrices and `cases`

Last time we saw that LaTeX has environments for making aligned equations, the `align` and `align*` environments. In this section we'll describe two other types of aligned environments in LaTeX, for making matrices and for making case-by-case definitions.

$$\begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \qquad f(x) = \begin{cases} 1, & \text{if } x \in \mathbb{Q}, \\ 0, & \text{if } x \in \mathbb{R} \setminus \mathbb{Q}. \end{cases}$$

[*]`danielericparker@gmail.com`

[†]`david_schwein@brown.edu`

1.1.1 Matrices

Here is an example of a matrix made with LaTeX.

```
\[
\begin{pmatrix}
a & b\\
c & d
\end{pmatrix}
\]
```

$$\begin{pmatrix} a & b\\ c & d \end{pmatrix}$$

What does this syntax mean? Every time LaTeX sees `&` it advances to the next column, and every time LaTeX sees `\\` it advances to a new row. The environment `pmatrix` makes matrices enclosed in parentheses. Other matrix environments include `matrix` (no parentheses), `bmatrix` (for `[ ]`), and `vmatrix` (for `| |`).

1.1.2 Dots

Typesetting an $n \times n$ matrix requires vertical, horizontal, and downward-sloped ellipses, as in the example at the beginning of this section.

`\ldots`	`\cdots`	`\vdots`	`\ddots`
$\ldots$	$\cdots$	$\vdots$	$\ddots$

The distinction between `\ldots` and `\cdots` is subtle: `\ldots` aligns the ellipsis with the bottom of the text while `\cdots` centers the ellipsis. Typographical style considerations dictate which of the two commands to use. For matrices and binary operations, use `\cdots`. For lists, use `\ldots`.

`x_1,\,x_2,\,\ldots,\,x_n`	`x_1 + x_2 + \cdots + x_n`
$x_1,\,x_2,\,\ldots,\,x_n$	$x_1 + x_2 + \cdots + x_n$

The command `\,` adds a little space after each comma. We'll talk more about it later.

1.2 `cases`

Here is an example of a case-by-case definition, written in LaTeX.

```
\[
|x| =
\begin{cases}
\phantom{-}x,
 & \text{if } x\geq 0,\\
-x, & \text{if } x<0.
\end{cases}
\]
```

$$|x| = \begin{cases} x, & \text{if } x\geq 0,\\ -x, & \text{if } x<0. \end{cases}$$

If you understand matrices you should understand the `cases` environment perfectly well: syntactically, the `cases` environment is a matrix that has exactly two rows.

A few other commands require explanation here. The command `\phantom{-}` adds an invisible $-$ to make the spacing come out right. (Try removing `\phantom{-}` and see what comes out.) The command `\text{...}` typesets its argument as regular text. Why did we add a space after `if`?

1.3 Accents and Font Styles in Math Mode

Some formulas require accents or special font styles. For example, mathematicians often indicate that the variable v stands for a vector by bolding it – like $\mathbf{v}$ – or by putting an arrow over it – like $\vec{v}$. The table below gives common accents and styles.

`\mathbf{v}`	`\mathbb C`	`\mathcal S`	`\mathfrak B`
$\mathbf{v}$	$\mathbb{C}$	$\mathcal{S}$	$\mathfrak{B}$
`\dot y`	`\ddot y`	`\overline{a+b}`	`\vec v`
$\dot{y}$	$\ddot{y}$	$\overline{a+b}$	$\vec{v}$
`\hat x`	`\widehat{x+y}`	`\tilde x`	`\widetilde{x+y}`
$\hat{x}$	$\widehat{x+y}$	$\tilde{x}$	$\widetilde{x+y}$

One unfortunate problem with these is that `mathbf` does not work with Greek characters. To fix this, load the `bm` package — the bold math package — which gives the command `\bm{ }`. This command works in math mode, and makes all of the enclosed symbols display as bold text.

1.4 Spacing

LaTeX allows fine control over spacing in math mode.

`|\!|`	`|\,|`	`|\>|`	`|\;|`	`|\quad|`	`|\qquad|`	`|\phantom{a+b}|`
$\|\!\|$	$\|\,\|$	$\|\>\|$	$\|\;\|$	$\|\quad\|$	$\|\qquad\|$	$\|\|$

The command `\!` adds *negative* space. Use it if LaTeX adds too much space to your formula and you want to tighten it.

`x^2/2`	$x^2/2$
`x^2\!/2`	$x^2\!/2$

The command `\phantom{ }` adds as much white space as its argument takes up. We already encountered it in the section on the `cases` environment.

Most of the spacing changes you'll make with these commands are subtle, yet mark the difference between nice math and ugly math. The more time you spend typing math and reading it, the better an eye you'll develop for these sorts of spacing questions. In the meantime, we recommend placing `\,` before dx in integrals. Compare an integral with `\,` with one without it.

```
\int_0^1 f(x)g(x)dx
```
$\int_0^1 f(x)g(x)dx$

```
\int_0^1 f(x)g(x)\,dx
```
$\int_0^1 f(x)g(x)\,dx$

1.5 Equation Numbering

It is often useful to number equations that are particularly important so that they may be referred to later.

```
\begin{equation}
x^n + y^n \neq z^n
\label{fermat}
\end{equation}
```

$$x^n + y^n \neq z^n \qquad (1)$$

The command `\label{fermat}` tells LaTeX that the name of the equation is `fermat`. We can reference it in the text using the command `\eqref`.

```
I proved \eqref{fermat},
but the margin
was too small
to contain it.
```

I proved (1), but the margin was too small to contain it.

To number a series of aligned equations, use the `align` environment. If you do not want to number a specific line in a series of aligned equations, place the command `\nonumber` on that line.

```
\begin{align}
z^n &= x^n + y^n
\nonumber\\
&\neq (x+y)^n
\end{align}
```

$$\begin{aligned} z^n &= x^n + y^n \\ &\neq (x+y)^n \qquad (2) \end{aligned}$$

1.6 Delimiters

Some expressions are so large that placing them in standard-size parentheses would look awful, like the following.

```
\[
(\int_0^1 f(x)\,dx)^2
\]
```

$$(\int_0^1 f(x)\,dx)^2$$

To make the sizes come out right, use `\left` before the left parenthesis and `\right` before the right parenthesis.

```
\[
\left(\int_0^1 f(x)\,dx\right)^2
\]
```

$$\left(\int_0^1 f(x)\,dx\right)^2$$

The `\left` and `\right` commands also work for other delimiters.

```
\[
\left\|\frac{x}{y}\right\|
\]
```

$$\left\|\frac{x}{y}\right\|$$

1.7 Making Math Operators

LaTeX comes with pre-defined commands for the most commonly used math operators, such as log and sin. But sometimes you may need to use an operator that has not already been defined. The proper way to do this is with the `\operatorname` command.

```
The subspace $A = \operatorname{span}\{x_1,x_2\}$
```

The subspace $A = \operatorname{span}\{x_1, x_2\}$

2 Making Custom Commands

Custom commands are one of the coolest and most useful features of LaTeX: they are commands that allow you to define other commands! They are very useful time-savers, especially if you find yourself having to type out similar or identical things many times. They are, however, a little bit tricky.

2.1 Custom Commands without Arguments

Let's look at a basic example and then go through it very carefully to see how it works. To start, let's define a command with no arguments.

```
\newcommand{\ftc}{Fundamental Theorem of Calculus}
\ftc

We can then apply the \ftc{} and conclude that...
```

Fundamental Theorem of Calculus
We can then apply the Fundamental Theorem of Calculus and conclude that...

What's going on here? The first line says we want to define a new command whose *name* is `\ftc` and, when that function is used, it should display "Fundamental Theorem of Calculus". Another way to think of this is in terms of copying and pasting. Typing a `\newcommand{\A}{B}` is like saying: copy `B` to memory and, every time you see `\A`, paste in `B`. You can define a `\newcommand` anywhere in the document. However, it will only work *after* the place where you have defined it. It's therefore good style to put any `\newcommand` in the preamble of the document. Also, one cannot make commands that already have names. If I try to make a newcommand called, say, `\textit`, then an error message will result because this command already exists. There are also some subtle rules about naming commands. As a rule of thumb, user-defined commands should contain only alphabetic characters — numbers and symbols don't work in command names properly.

It's also important to note that, even though our command has no arguments, one should still include empty curly braces after it. Why is this. Try typing the following instead:

```
\newcommand{\ftc}{Fundamental Theorem of Calculus}
\ftc

We can then apply the \ftc and conclude that...
```

Fundamental Theorem of Calculus
We can then apply the Fundamental Theorem of Calculusand conclude that...

As you can see, without the braces, the space is missing after the command. This is because LaTeXassumes that the next character after a command is its first argument, even if the command has no arguments. By explicitly giving the function an empty argument, we can avoid this behavior.

2.2 Custom Commands with Arguments

User-defined commands can have multiple arguments, just like normal commands. `\newcommand` has an optional argument that says how many arguments defined command should have. Suppose we were doing a problem which involved integrating a lot of functions of x over the entire real line. Then it might come in handy to have a command that made an integral from $-\infty$ to ∞ that integrated with respect to x. Again, it's easiest to see how this works by example.

```
\newcommand{\intx}[1]{\int_{-\infty}^\infty #1 \, dx}
\[
\intx{f(x)}
\]
\[
\intx{\frac{3x}{5}e^{-3x}}
\]
```

$$\int_{-\infty}^\infty f(x) \, dx$$

$$\int_{-\infty}^\infty \frac{3x}{5}e^{-3x} \, dx$$

Even time LaTeXsees a `#1`, it replaces it with the first argument given when the defined command is used. This can appear multiple times inside the command. For instance, we can make a command that repeats something three times:

```
\newcommand{\rep}[1]{#1 #1 #1}
\rep{a} \rep{eggplant}
```

a a a eggplant eggplant eggplant

Making a command that has more than one argument can be done in the same way.

```
\newcommand{\comparison}[2]{I think that #1 is \textit{way} better than #2.}
\comparison{Pepsi}{Coke}
\comparison{Star Trek}{Star Wars}
```

I think that Pepsi is *way* better than Coke.
I think that Star Trek is *way* better than Star Wars.

Hopefully by now you are convinced how cool and how useful macros are. Used carefully and thoughtfully, they can save a great deal of time when writing it LaTeX. If you need to type the same, or a very similar, thing more than three or four times, then it may be worth making it into a macro. Over time, one tends to accumulate a collection of standard macros that you prefer to use. Many people have a few dozen `\newcommand`'s that they put in the preamble of each LaTeX file they make.

3 Tables

Making a table in LaTeX is very similar to making a matrix, with a few slight modifications.

```
\usepackage{booktabs}
...
\begin{center}
\begin{tabular}{ l r }
\toprule
City & Population \\
\midrule
Providence & 178,024 \\
Warwick    &  82,672 \\
Cranston   &  80,387 \\
Pawtucket  &  71,148 \\
\bottomrule
\end{tabular}
\end{center}
```

City	Population
Providence	178,024
Warwick	82,672
Cranston	80,387
Pawtucket	71,148

Let's break down this example. We first load the package `booktabs`, for making typographically good tables. Our table consists of two environments: the `center` environment, which centers the table on the page, and the `tabular` environment, which makes the table proper. The `tabular` environment takes a second argument, in our case `l r`, which tells LaTeX how many columns the table will have and how to align them. The letter `l` is for left-alignment, the letter `r` is for right-alignment, and the letter `c` is for centering. Finally, the commands `\toprule`, `\midrule`, and `\bottomrule` tell LaTeX where to place the horizontal lines separating the descriptors and the columns.

Tables can be very complicated, and for this reason there are many LaTeX packages that provide special table functionality, such as the `colortbl` environment for colored tables. See the LaTeX wikibook for a complete list.

There is a second syntax for making tables, which uses `\hrule` for making horizontal lines and uses vertical bars (the character `|`) in the second argument of the `tabular` environment for making horizontal lines.

```
\usepackage{booktabs}
...
\begin{center}
\begin{tabular}{|l||r|}
\toprule
City & Population \\
\midrule
Providence & 178,024 \\
Warwick    &  82,672 \\
Cranston   &  80,387 \\
Pawtucket  &  71,148 \\
\bottomrule
\end{tabular}
\end{center}
```

City	Population
Providence	178,024
Warwick	82,672
Cranston	80,387
Pawtucket	71,148

It's good to know the second syntax because it is a standard LaTeX feature. We told you about making tables with `booktabs` first because those tables are of much greater quality.[1]

4 Miscellany

4.1 Adjusting Margin Sizes

The default margins in LaTeX documents are quite wide, and while there is good typographical justification for their size, many people prefer to decrease the margin width. If you are one of these people, we recommend using either of the following packages: the `fullpage` package, which automatically adjusts the margins, or the `geometry` package, which allows finer control over the margins.

`\usepackage{fullpage}` or `\usepackage[margin=2cm]{geometry}`

4.2 Optional Arguments for `\documentclass`

When we told you about the `\documentclass` command, we left out some functionality: `\documentclass` takes optional arguments. For example, writing `\documentclass[twocolumn,12pt]{article}` at the beginning of your TeX file will typeset the document in two columns at 12-point font. See the LaTeX wikibook chapter on document structure for a complete list of optional parameters.

[1]Specifically, tables made with `booktabs` are better because they have no vertical lines and few horizontal lines. These characteristics are hallmarks of professional table design.

5 Bibliography Management and BibTeX

BibTeX is a slight extension to LaTeX for managing references. It's made to be easy to use and highly convenient. Citations are references are generally bothersome to do by hand, for two reasons:

1. Everyone wants a very specific, but slightly different format for citations (e.g. MLA, Chicago, etc).
2. Citations are often numbered by order of appearance, so any time you insert a new citation, all the numbers change.

BibTeX has two elegent solutions to these problems:

1. Citation formatting is automatic.
2. Citation numbering is automatic.

This is one of the best examples of the LaTeXphilosophy; BibTeX does all the difficult formatting for you, so you can focus on content. There are three steps to using BibTeX: creating a bibliography file, citing with the `\cite` command, and typesetting for BibTeX. Let's look at them in order.

5.1 Creating a Bibiography File

BibTeX requires an external bibliography. There are several external tools to create these quickly, but it's useful to understand how they work before taking shortcuts. This takes the form of a file in the same folder as your `.tex` file that has the extension `.bib`. It's customary to call it `references.bib` or `citations.bib`, or something along those lines. The `.bib` files contains the information about each thing you want to cite.

BibTeX supports many types of bibliographic entries, including `book`, (journal) `article`, (conference) `proceedings` and many more. Let's look at a few examples:

```
@book{rudin1964principles,
  title={Principles of mathematical analysis},
  author={Walter Rudin},
  volume={3},
  year={1964},
  publisher={McGraw-Hill New York}
}

@article{bardeen1957theory,
  title={Theory of superconductivity},
  author={Bardeen, John and Cooper, Leon N and Schrieffer, J Robert},
  journal={Physical Review},
  volume={108},
  number={5},
  pages={1175},
  year={1957},
  publisher={APS}
}
```

Each entry starts of with and then the name of that type of entry. This is followed by the *name* of that entry, a unique identifier that you can use to cite that entry. For the second entry above, the name is `bardeen1957theory`. After that, there several different fields that give the information needed to construct the reference. For `book`, the required fields are `title, author, publisher, year`. For `article`, it's also necessary to include `journal`, and highly recommended to provide `year, volume, page` so that a reader could easily find the article in question. A list of entry types and what information is required for each one is available at `http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management`. Notice that there is a *comma* after each field. This is absolutely necessary to include.

Of course, no one wants to bother typing all this out for each citation, so let's look at easier ways to do this. Luckily, there are many good ones!

Google Scholar If you find your reference on Google Scholar, then you get the BibTeX entry for free! Under the name of the entry in Google Scholar, there are several links: "Cited by...", "Related Articles", "Cite" and several more. If you click "cite", you will get a citation for that work in several formats, one of which is BibTeX. These days, most journals have all their articles online and have a similar "generate citation" button for each article.

JabRef A (free, open source) program for managing BibTeX files. Works on Windows, Macs, and Linux. Gives a nice graphical user interface (GUI) for adding entries. It also provides an interface to search online journal databases and turn the search results into bibliography entries.

BibDesk A similar tool, that is slightly more polished and user friendly, but only works on Macs. This is included with TeXShop.

5.2 Citing in the Text

Citing from the text is very easy. Let's look at an example.

```
A standard textbook on Analysis is \cite{rudin1964principles}.

The theory of low-temperature
superconductivity was invented in 1957 \cite{bardeen1957theory}.

\bibliographystyle{plain}
\bibliography{references.bib}
```

A standard textbook on Analysis is [2].
The theory of low-temperature superconductivity was invented in 1957 [1].

References

[1] John Bardeen, Leon N Cooper, and J Robert Schrieffer. Theory of superconductivity. *Physical Review*, 108(5):1175, 1957.

[2] Walter Rudin. *Principles of mathematical analysis*, volume 3. McGraw-Hill New York, 1964.

To cite a bibliographic entry, use the `\cite` command with the name of that entry. The command `\bibliographystyle{plain}` says how the citations should be formatted. The `plain` style will sort references by author and use squres brackets with numbers in the text. There are many other styles available, such as `unsrt`, which gives the citations in order of appearance. The `alpha` style will give citations like `[Rud64]`. The `natbib` package gives many more options, including `apa`. To use MLA-style citations, try the `biblatex-mla` package, which is a little more involved. Many many other formats are available, and can be found with a little googling around.

5.3 Typesetting with BibTex

Since BibTeX has an external file, you need to typeset a little differently. After any changes to the BibTeX file, you must go through the following sequence:

1. Typset LaTeX normally.
2. Typeset BibTeX. This is normally a separate button or menu option.
3. Typset LaTeX normally.
4. Typset LaTeX normally *again*.

5.4 URLs and DOIs In Citations

Almost all journals are online now, so it's usually more useful to give the DOI or a link to an article than the journal issues. To do this, load the `hyperref` and `doi` packages, and add the `url` and `doi` packages to your bibtex entries. As an example:

```
@article{bardeen1957theory,
  title={Theory of superconductivity},
  author={Bardeen, John and Cooper, Leon N and Schrieffer, J Robert},
  journal={Physical Review},
  volume={108},
  number={5},
  pages={1175},
  year={1957},
  publisher={APS}
  doi = {10.1103/PhysRev.108.1175},
  url = {http://link.aps.org/doi/10.1103/PhysRev.108.1175}
}
```

In some — but not all — bibliography styles, these will show up at the end of the citation. Try the `plainnat` style to get this to work (requires the `natbib` package).

6 The `siunitx` Package

`siunitx` is a package for typesetting units. Units are something of a pain to typeset in normal LaTeX, for a few reasons. Suppose we had measured the frequency of an oscillation to be 3.0×10^{-6}Hz. Using math mode this could be written as `$3.0 \times 10^{-6} \text{Hz}$`. Not only is this long, but the spacing between the number and the unit is terrible! The `siunitx` package offers a better way to do this. Load the package with the command `\usepackage{siunitx}`

`\SI{3e-6}{\hertz}`	$3 \times 10^{-6}\,\mathrm{Hz}$

This is much shorter and faster. The `siunitx` package supports all SI units, as well as quite a few others. It also supports standard prefixes, such as giga- and femto-. Let's look at a few more complex examples.

`\SI{3}{\giga\hertz}`	3 GHz
`\SI{3}{GHz}`	3 GHz
`\SI{19}{\meter\per\second}`	$19\,\mathrm{m\,s^{-1}}$
`\SI[per-mode=fraction]{19}{\meter\per\second}`	$19\,\frac{\mathrm{m}}{\mathrm{s}}$
`\SI{54.1}{kg.m/s^2}`	$54.1\,\mathrm{kg\,m/s^2}$
`\SI{32932e-3213}{\micro F}`	$32\,932 \times 10^{-3213}\,\mathrm{\mu F}$
`\SI{3.2}{kg/\nano\meter\squared}`	$3.2\,\mathrm{kg/nm^2}$
`\SI{8.9}{\kilo\gram\tothe{12}}`	$8.9\,\mathrm{kg^{12}}$
`\SI{27}{\degreeCelsius}`	27 °C

There are lots of things going on here. First, you can use either the standard names for units, which all have their own commands, like `\hertz`, or simply type `Hz`. By default, fractions are displayed as negative powers, but you can modify this with the optional argument `[per-mode=fraction]`. If you import the package with the command `\usepackage[per-mode=fraction]{siunitx}`, then all units will be displayed as fractions by default. Lastly, we get access to many base-10 prefixes: kilo-, mega-, giga-, tera-, centi-, milli-, micro-, nano-, pico-, and many more, from 10^{-24} to 10^{24}.

The `siunitx` package contains several other useful commands. `num` works just like the first argument of `\SI` by itself, and `\si` works like the second part.

`\num{3.0}`	3.0
`\num{3.0e5}`	3.0×10^5
`\num{3+2i}`	$3 + 2\mathrm{i}$
`\si{\kilo\gram\per\tesla}`	$\mathrm{kg\,T^{-1}}$
`\si{\lumen\per\radian}`	$\mathrm{lm\,rad^{-1}}$

Additionally, there is a special command for typing numbers with the same units. Numbers are separated by semi-colons, and commands and an "and" are inserted automatically.

```
\SIlist{1;2.3;-534;7.3e-5}{\joule\per\second}
```
$1\,\mathrm{J\,s^{-1}}$, $2.3\,\mathrm{J\,s^{-1}}$, $-534\,\mathrm{J\,s^{-1}}$ and $7.3 \times 10^{-5}\,\mathrm{J\,s^{-1}}$

For more information on this package, check out it's manual on CTAN: `http://www.ctan.org/pkg/siunitx`.

7 The `mhchem` Package

The `mhchem` package serves a similar function with chemical equations. LaTeX's math mode is meant to write mathematics and is thus rather ill-suited for typing chemical equations, mostly because the atomic symbols should not be italicized, and it's a pain to break out of italics all the time. To load this package, use the command `\usepackage{mhchem}`. This allows access to a new command `\ce`, which stands for *chemical equation*. Again, some examples are probably the best way to demonstrate this.

`\ce{H2O}`	$\mathrm{H_2O}$
`\ce{H2SO4}`	$\mathrm{H_2SO_4}$
`\ce{NO3-}`	$\mathrm{NO_3{}^-}$
`\ce{AgCl2-}`	$\mathrm{AgCl_2{}^-}$
`\ce{CrO4^2-}`	$\mathrm{CrO_4{}^{2-}}$
`\ce{^{235}_{92}U}`	${}^{235}_{92}\mathrm{U}$

Not only is it possible to do chemical compounds, but also entire chemical equations.

`\ce{CO2+C -> 2CO}\\[0.4em]`	$\mathrm{CO_2{}^+C \longrightarrow 2\,CO}$
`\ce{H+ + OH- <=>> H2O}`	$\mathrm{H^+ + OH^- \rightleftharpoons H_{20}}$

This package contains many many more options and variations on this. If you want to learn more, head to the manual at: `http://www.ctan.org/pkg/mhchem`.

8 The `amsthm` Package

Professional mathematics publications often separate theorems, definitions, and other important information from the text, like so.

Theorem 1. *If $n \geq 3$, the equation $x^n + y^n = z^n$ has no integer solutions.*

Doing this in LaTeX requires the `amsthm` package. To make a theorem, you have to first tell `amsthm` to define a new environment. The command to use is `\newtheorem`, which takes two arguments. The first is the name of the new environment, and the second is what should be displayed. For instance, placing `\newtheorem{theorem}{Theorem}` in the preamble allows us to make a theorem using the newly defined `theorem` environment.

```
\newtheorem{theorem}{Theorem}
...
\begin{theorem}
Every finite $p$-group
has nontrivial center.
\end{theorem}
```

Theorem 2. *Every finite p-group has nontrivial center.*

To prevent LaTeX from numbering the environment, use the `\newtheorem*` command.

```
\newtheorem*{proposition}{Proposition}
...
\begin{proposition}
Every finite $p$-group
has nontrivial center.
\end{proposition}
```

Proposition. *Every finite p-group has nontrivial center.*

Every environment that is created in this way takes an *optional argument.* We haven't talked about optional arguments so far, but the basic concept is very simple. An optional argument for a command, if there is one, must be placed in square brackets `[ ]`. For example, the square root command `\sqrt` takes an optional argument indicating the degree of the radical.

`\sqrt[n]{x^2+1}` $\sqrt[n]{x^2+1}$.

The optional argument for an `amsthm` environment indicates text to be placed in parentheses.

```
\begin{theorem}[Feit-Thompson]
Every group
of odd order
is solvable.
\end{theorem}
```

Theorem 3 (Feit-Thompson)**.** *Every group of odd order is solvable.*

8.1 Proofs

The `amsthm` package does come with one environment predefined, the `proof` environment, which does just what you expect.

```
\begin{proof}
A trivial exercise
for the reader.
\end{proof}
```

Proof. A trivial exercise for the reader. □

If you end a proof with a displayed equation, the Halmos box[2] may not go in the right position. To place it in a specific spot, use the command `\qedhere`.

```
\begin{proof}
Just observe that
\[
1 + 1 = 2.
\]
\end{proof}
```

Proof. Just observe that

$$1 + 1 = 2.$$

□

```
\begin{proof}
Just observe that
\[
1 + 1 = 2.
\qedhere
\]
\end{proof}
```

Proof. Just observe that

$$1 + 1 = 2. \qquad \square$$

8.2 Theorem Styles

There are three possible styles for environment defined using `amsthm`: namely `plain` (the default), `definition`, and `remark`. To specify a style, group `\newtheorem` commands into blocks according to the style you would like them to have, then place `\theoremstyle{<style>}` before each block. For instance, part of your preamble might look like

```
\theoremstyle{theorem}
\newtheorem{theorem}{Theorem}
\newtheorem{proposition}{Proposition}
\newtheorem{lemma}{Lemma}

\theoremstyle{definition}
\newtheorem{definition}{Definition}
\newtheorem{exercise}{Exercise}

\theoremstyle{remark}
\newtheorem*{remark}{Remark}
```

[2]The Halmos box is the symbol □.

9 The TikZ Package

Ti*k*Z is a package for TeX/LaTeX for drawing figures, diagrams, and images.[3] It's comprehensive, and extremely powerful, and capable of doing pretty much anything you could want it to. My goal for this next hour is to show you some of the basics of Ti*k*Z, so that you can get started making figures. Before I begin, I'll try and convince you that's its worth knowing, by showing you some of the figures you can create with it.

Ok, let's get started.

How do you use Ti*k*Z?

First, include the package: `\usepackage{tikz}`.

Now, to actually create a figure, begin the environment '`tikzpicture`'. If you're going to be putting this figure into a larger document, you'll probably want to surround with a 'figure' environment so that it floats appropriately, and so you can give it a caption and label.

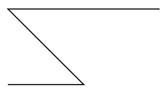
One thing that makes Ti*k*Z a little confusing at first is that it has an entirely different syntax than LaTeX, and the Ti*k*Z code you'll be writing won't necessarily look like anything you've seen in LaTeX before.

Once you have the environment set up, you start drawing. The basic way to draw something in Ti*k*Z is with a command like:

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

Commands will usually start with '`\draw`' and then have a series of coordinates with instructions between them, ending with a semicolon. Ti*k*Z has a nice shorthand for creating pictures with a single command. I can write '`\tikz \draw ...;`' instead of beginning and ending an environment. We'll use this shorthand to spare typing whenever possible. Here's an example:

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



and is exactly the same as

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

You can read the previous command as saying 'Draw a path starting at (0, 0) then connected by a straight line to (1, 0) then connect that with a straight line to (1, 0)', and so on . . .

The coordinates here are in an implicit cartesian plane, as familiar from high school algebra and calculus. The only difference is that the grid is implicit, and not displayed with your figure.

[3]This section was prepared with the help of Spencer Gordon.

Two interesting things to observe: First, the default coordinate system is has the positive x-axis pointing right, and the positive y-axis pointing up, and one unit corresponds to one centimeter. If you want to work with different units, you can simply add the units to your coordinates, like so: '`(2in,0.3pt)`'. It's pretty straightforward to resize everything after you've made an image, so it's probably a good idea to ignore absolute sizes of things when creating an image, and only afterward, try to make things fit in a certain space.

Second, Ti*k*Z takes care of computing the size of your figure and making sure that your entire figure is visible in the '`tikzpicture`' so you also don't have to worry about any part of your figure getting cut off.

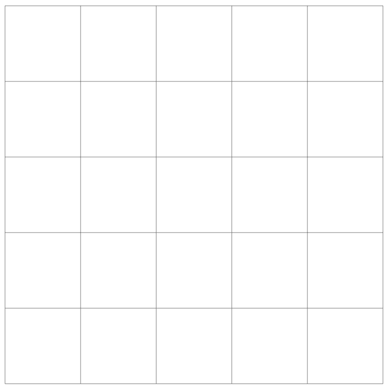
Finally, for those of you who don't have experience with computer graphics programming, you'll have to get used to thinking in terms of coordinates when creating your diagrams, which might be a challenge at first, but Ti*k*Z can help you out.

Another available command that will almost certainly be useful when planning out a figure is '`\grid`'.

Here's an example:

```
\tikz \draw[help lines] (0, 0) grid (5, 5);
```

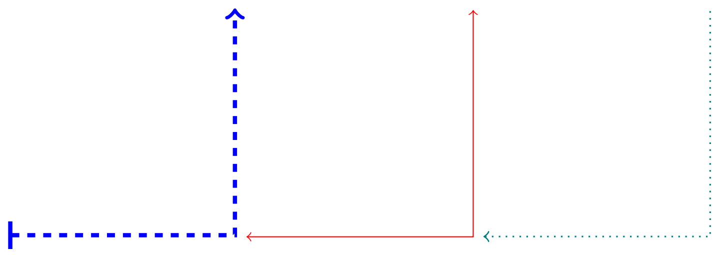
Let's take a look at what this generates.



There are two differences between this draw command and the previous one. In the previous one, we used the '`--`' instruction to say draw a straight line between the previous and following coordinates. Here we are using the '`grid`' instruction to draw a grid in the rectangle with corners at $(0, 0)$ and $(5, 5)$.

The other new feature is the option given to the draw command, '`help lines`'. This is an example of a *style*, which is a collection of different options, each of which controls a parameter used when drawing, such as line width, line color, line style, coordinate transformations, line start and end decorations, and many many more. See the manual (available online for the full list.) A style is just a collection of settings that can be invoked at once. In this case, '`help lines`' just sets the line width to thin, and makes the line color a lighter gray.

To give you a sense of the possibilities available, the following figures all have the very same path drawn, just with different options.

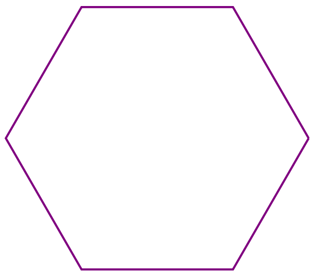


I'll let you play around with what you've seen in a second, but I just want to present two alternative ways of specifying coordinates.

First, polar coordinates. For example, I can write

```
\tikz \draw[color=violet, thick] (0:2) -- (60:2) -- (120:2) -- (180:2)
  -- (240:2) -- (300:2) -- (0:2);
```

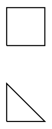
to draw a hexagon using polar coordinates. To use polar coordinates, you specify an angle (in degrees) followed by a `:`, followed by a radius. You can freely mix polar and cartesian coordinates in your figures.



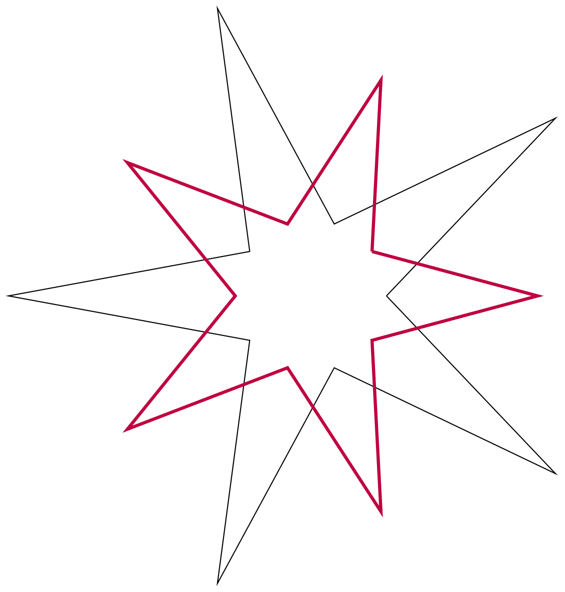
Finally, you can specify relative coordinates with a neat special syntax. If you want to say 'the point 2 units above the previous point, you can write: '`++(2, 0)`'. Here's an example. Let's say I want to draw a square 3 units above a triangle. Instead of translating all the coordinates in my head when drawing the square, I can just do the following:

```
\begin{tikzpicture}[scale=0.5]
\draw (0, 0) -- (0, 1) -- (1, 0) -- (0, 0);
\draw (0, 2) -> ++(1, 0) -> ++(0, 1) -> ++(-1, 0) -> ++(0, -1);
\end{tikzpicture}
```

This makes the figure



Ok, now an exercise. I would like you to try and recreate this figure:



Now that you have a decent grasp on the basic functionality needed to make line drawings, we'll move on to some more sophisticated functionality.

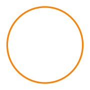
I can draw a curve as follows:

```
\tikz[scale=0.4] \draw (0, 0) to [out=135, in=100] (1, 1);
```

The `out` option specifies the angle at which the curve should leave the starting point and the `in` option specifies the angle at which the curve should arrive at the end point.

To draw a circle,

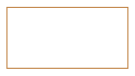
```
\tikz \draw[orange, semithick] (1, 1) circle [radius=0.5];
```



The first point is the center in this case.

To draw a rectangle,

```
\tikz[scale=0.4] \draw[brown] (0, 0) rectangle (4, 2);
```



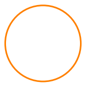
Here the first point is one corner of the rectangle and the second point is the opposite corner of the rectangle.

There are many many more handy shortcuts that Ti*k*Z provides for common shapes and patterns, and you should take a look at the manual to find out more, but I would like to move on.

You may not be surprised at this point to learn that the draw command I've been using this whole time is actually shorthand for yet another command, which is the actual command underlying all these different path commands, namely '`\path`'.

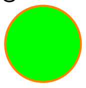
The command I used for the circle just a minute ago was equivalent to the following command, which is exactly what Ti*k*Z expands the above command into.

```
\tikz \path[draw=orange, semithick] (1, 1) circle [radius=0.5];
```



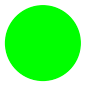
Just like there is a '`draw`' option to the `path` command, there is also an option to `fill` a path, which can be given in the same way, so I can fill the same circle as above with:

```
\tikz \path[draw=orange, fill=green, thick] (1, 1) circle [radius=0.5];
```



I'm actually both filling and drawing this circle, and with two different colors. I can choose to just fill the circle by using a shortcut

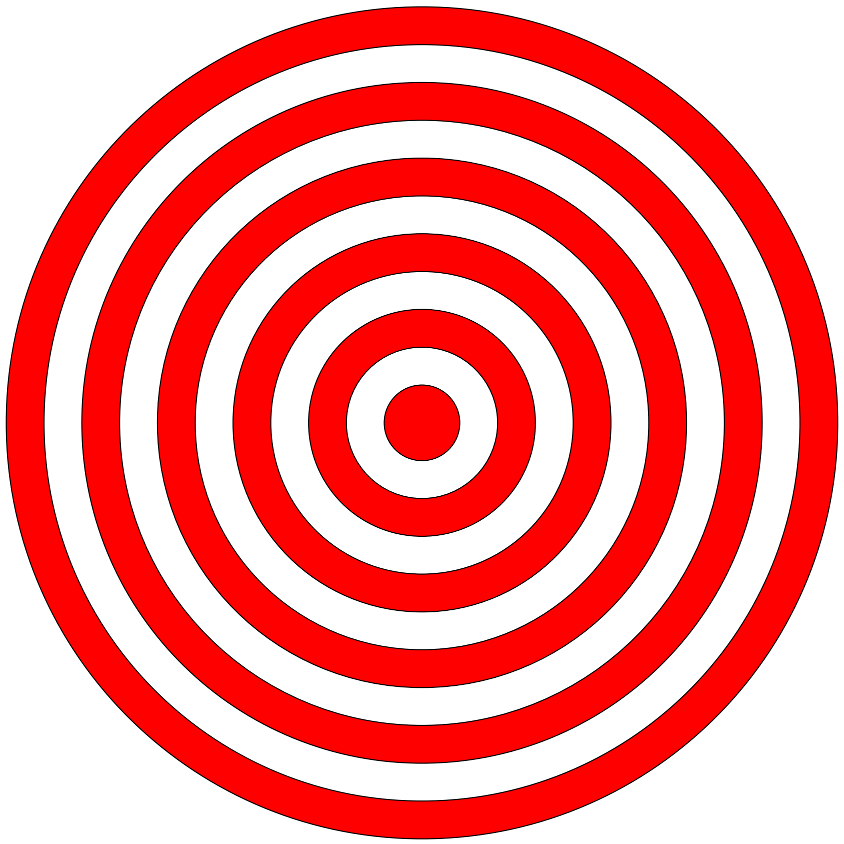
```
\tikz \fill[green] (1, 1) circle [radius=0.5];
```



or by using the '`none`' value instead of color as the value for the '`draw`' option.

Rather than go through any more small isolated pieces of functionality, I'm going to try and work through a larger example, and introduce some new concepts as needed.

Here's the diagram I want to create:



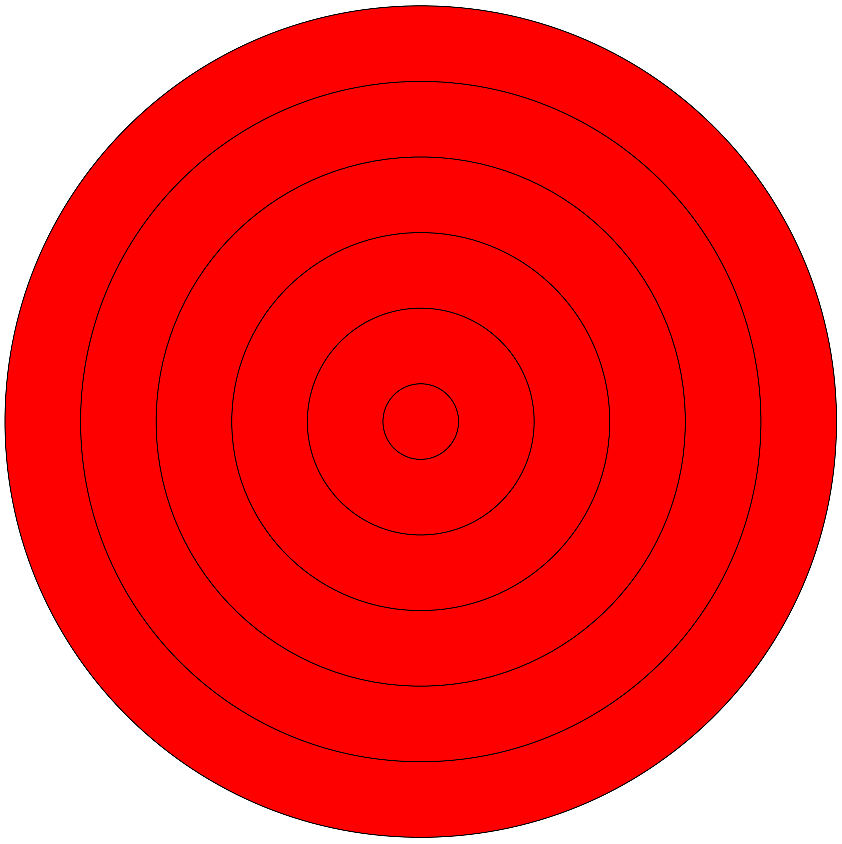
Looking at this, it seems clear that writing this by hand using the `\draw ... circle ...;` would be really annoying, and if I wanted to adjust it at all, I'd have to make a lot of changes so everything lined up. So Ti*k*Z offers a really nice feature, which will be familiar to those of you with some programming experience: For-loops. Here's the code I used to make this bullseye-like figure:

```
% Bullseye
\begin{tikzpicture}
\foreach \x in {5,4,...,0}{
  \draw[fill=red,draw=black] (0,0) circle[radius=\x+0.5];
  \draw[fill=white,draw=black] (0,0) circle[radius=\x];
};
\end{tikzpicture}
```

The interesting part is the loop, which begins with `\foreach \x in {5,4,...,0}`. This first part means, take whatever code I have below, and copy and paste it once for each item in the list going $5, 4, \ldots, 0$ and have `\x` be the current item in the list each time. Note that we used the value of `\x` in our for-loop to adjust the radius of each circle that we draw, and that we do simple arithmetic to make the red circle bigger than the white circle.

Here's a question for you: What would happen if we switched the order in which we drew the two circles?

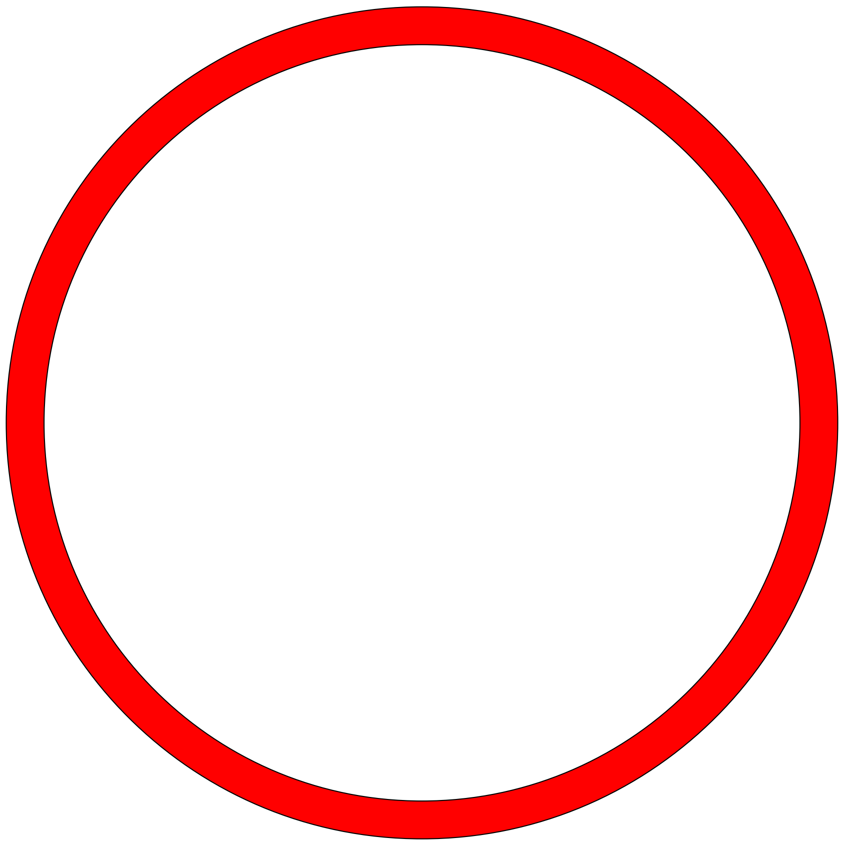
Here's the answer:



What happens is the red circles are drawn after the white circles, and go right on top of the white circles, so the white circles can't be seen.

Another question: What would happen if we went back to the original order for drawing the circles, but instead of counting down, counted up?

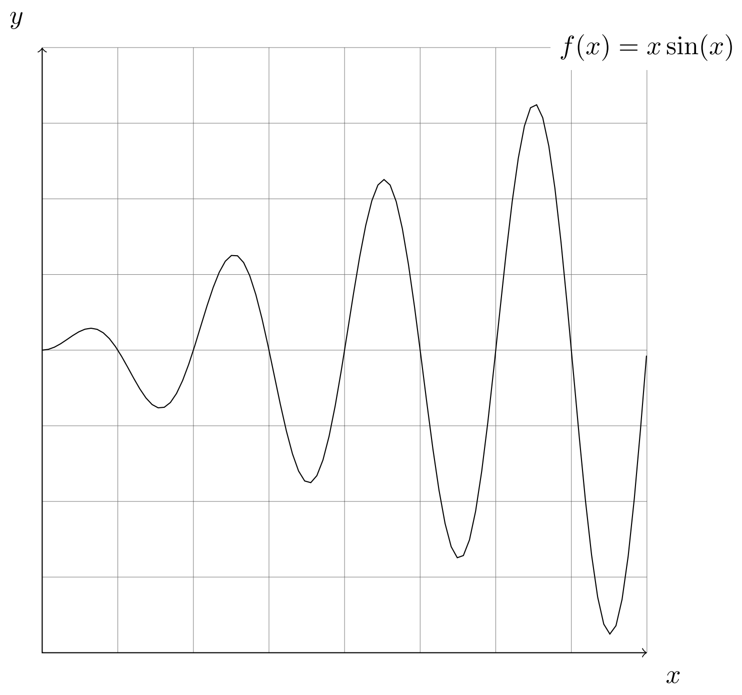
Here's the answer:



The last two circles cover all of the previous circles, so they can't be seen.

Basically, in Ti*k*Z, every new part of a drawing goes on top of everything else, so you need to make sure to pay attention to the order in which you draw things to ensure that they don't overwrite each other incorrectly. Sometimes, this is actually very handy, since you'll want to draw something on top of part of your drawing, and it's nice to not have to worry about specifying what's behind what, when all you want is newer stuff to be on top of older stuff.

Ok, I'll do one more example, and then give you exercises to work on. Here's what I want to produce:



I make this diagram with a few commands. First, I use

```
\node[label=above left:$y$] (topLeft) at (0, 2) {};
\node[label=below right:$x$] (botRight) at (2, 0) {};
```

This defines two new coordinates, which can be referred to by `(topLeft)` and `(botRight)` respectively, and adds labels to the image with positions relative to the coordinates. The `{}` contains any text associated with the node. In this case, we don't want the nodes themselves to have text.

Generally, nodes are like coordinates in that they have a location and can be referred to later, but they can also have their own appearance and text content, though I'm not using either of those features in this case.

To give a node a label, use the `label` option and provide a location relative to the node, and then the label text. I find it pretty confusing to talk about node labels, since nodes usually contain text themselves, and are often used as labels, but remember that labels are always associated with a node, while nodes can be inserted pretty much anywhere in a path.

Next, I draw the grid using the grid command, specifying that I want horizontal lines to be drawn at every `ystep` units, and vertical lines to be drawn at every `xstep` units, and that I want them to be drawn with very thin, gray lines.

Now, I draw the axes with a single path, starting from the center of the `topLeft` node, and going to the origin, then the center of the `botRight` node.

The next command actually draws the curve. For this to work, you'll need to add the command `\usetikzlibrary{calc}` to the preamble. This tells Ti*k*Z to load the `calc` library.

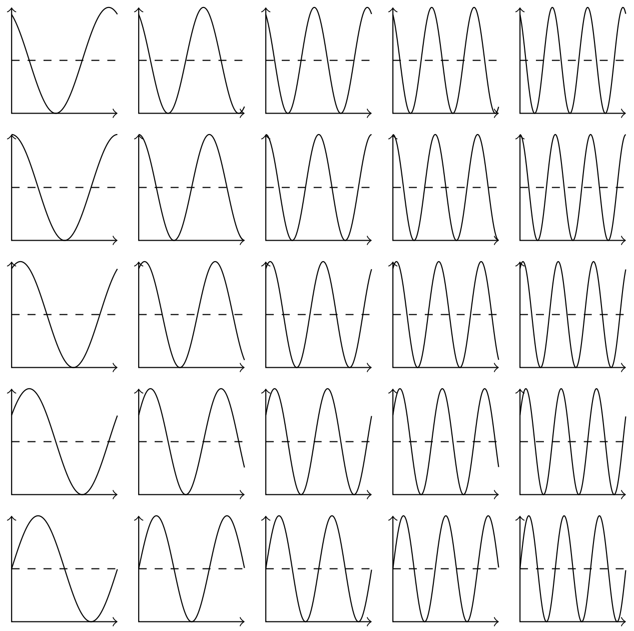
```
\draw[domain=0:2,fill=none,samples=100,draw=black]
  plot (\x, {\x*sin(2*360*\x)/2 + 1});
```

This command uses the plot path instruction, which creates a path from the plot of a function. I specify the function with `(\x, {\x*sin(2*360*\x)/2 + 1})`. Note that you have to use `\x` as your variable in the plot instruction unless you use an option to select a different variable name. Also, the function must be surrounded by curly braces, and can use any of many builtin functions. In this example we only use sine, but there are many builtins, which can be found in the documentation.

Two other things to note: We use the `domain` option to define the domain of coordinates over which we are plotting the function. If you want to plot a function over a domain that's different from the coordinates in which you putting the plot, you'll have to use some other options to shift things around, but this will work for the time being.

The way Ti*k*Z actually plots the function is by taking samples of the function value at various points in the domain and interpolating the missing values, and then trying to smooth out the curve, so the more complex your function the more samples you'll need to get something that looks right. In this case, I use the `samples` option to specify that Ti*k*Z should use 100 samples.

I don't have time to explain this in its entirety, but at this point you should be almost able to create some pretty cool complex figures like this.



10 Additional Resources

This set of notes is deliberately brief, which is good for getting across the important ideas but not good for communicating subtleties. If you would like to know more, the internet has extensive resources for learning and mastering LaTeX; the following are a few of our favorites.

- The Wikibooks LaTeX page (`http://en.wikibooks.org/wiki/LaTeX`). A compendium of basic and advanced information about LaTeX. If you want to know how to do something with LaTeX, this website can probably tell you how.
- Detexify (`http://detexify.kirelabs.org`). The LaTeX names for mathematical symbols can be hard to remember or look up because there are so many of them. Detexify lets you draw a symbol on the computer screen, then guesses which symbol you drew and tells you the LaTeX command for it.
- TeX Stack Exchange (`http://tex.stackexchange.com`). A question-and-answer site about TeX, LaTeX, and friends. The users of the site – among them people who designed LaTeX – are pros, and can answer about any question you have. Save it for the stumpers, though: Stack Exchange sites discourage users from asking a question whose answer can be easily looked up.
- The Comprehensive TeX Archive Network (CTAN) (`http://www.ctan.org`) Members of the LaTeX community have written over 4500 packages that add extra functionality to LaTeX. CTAN houses these packages and their documentation. When you have a question about a package, the documentation is often the best place to start.