1 *Mathematica* Basics

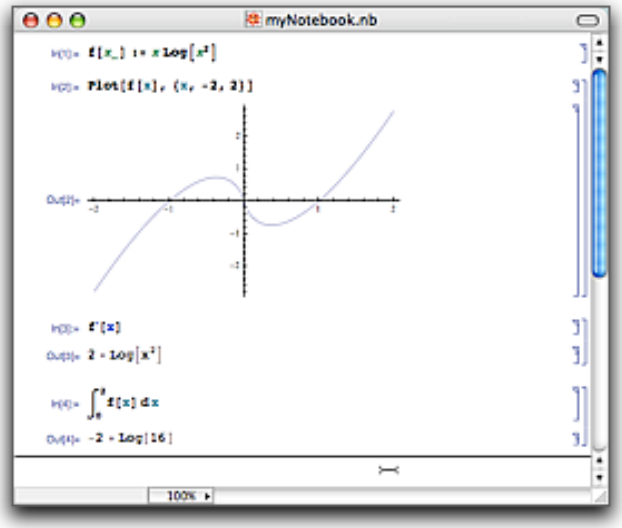
This chapter is an introduction to *Mathematica*. We briefly describe many of the most important and basic elements of *Mathematica* and discuss a few of the more common technical issues related to using *Mathematica*. Since our primary goal is to use *Mathematica* to help us understand calculus, you should not initially spend a great amount of time pouring over the details of this chapter, except as directed by your professor. Simply familiarize yourself with what's here, and refer back to it later as needed.

1.1 Getting Started

Any new user of *Mathematica* must understand several basic facts concerning the user interface, syntax, and the various types of objects that one encounters in using *Mathematica*. This section is a cursory look at some of these fundamentals.

■ The *Mathematica* "Front End"

When you start up *Mathematica*, the first thing you see is a window displaying the contents of a "**notebook**." This window is displayed by *Mathematica*'s **front end**. The front end is the interface between you and the *Mathematica* **kernel**, which does the computations. The following is a typical (simple) notebook in a front end window.



A *Mathematica* notebook is composed of **cells**. On the right side of the window you see **cell brackets**. Each cell in the notebook shown above is either an *input cell*, an *output cell*, or a *graphics cell*. There are several other kinds of cells. Some of these are *text*, *title*, and *section*.

Also notice the horizontal line near the bottom of the window. This indicates the insertion point for the next new cell. To enter a command into a notebook, simply begin typing. The default cell type is *input*. When you're done typing, just press **shift-return** (on a Macintosh, you can also use the "**enter**" key.) To evaluate an existing input cell, simply click anywhere inside the cell (or on the cell bracket) and press **shift-return** (or **enter**).

To create a cell *between* two existing cells, move your cursor over one of the cells toward the other until the "I-beam" cursor becomes horizontal. Then click, and a horizontal line will appear, indicating the desired insertion point. To delete a cell, click on its bracket and then choose Clear from the Edit menu or simply press the DEL key

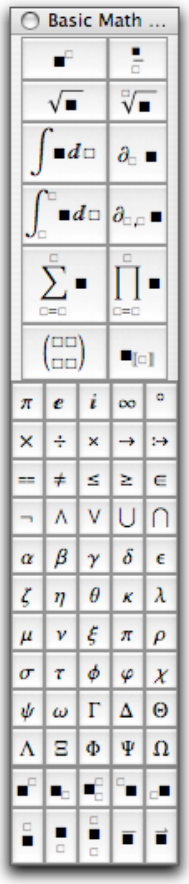
Palettes. You can enter mathematical expressions so that they appear essentially the same as you would write them on paper or see them in your textbook. For example, to define the function $f(x) = \sqrt{x^2 + 1}$, we could use the "Input Form"

```
f[x_] := Sqrt[x^2 + 1]
```

or we could use "Standard Form":

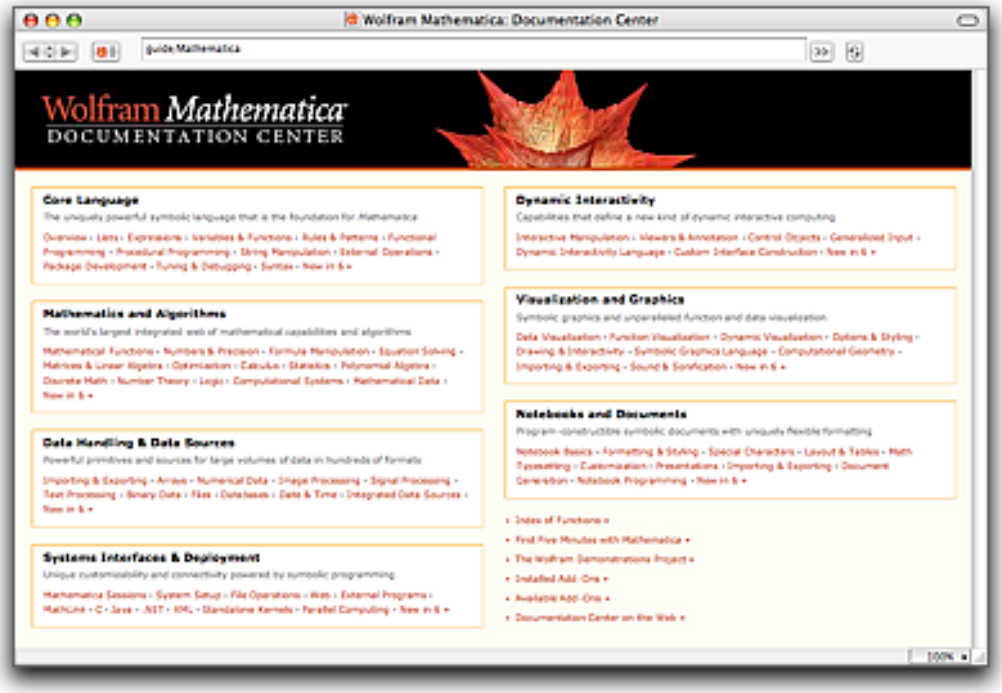
```
f[x_] := √(x^2 + 1)
```

There is a vast set of keystroke combinations for typing such expressions. However, at first, you will probably want to take advantage of one or more or the standard *palettes* that are available. The image to the right shows the BasicInput palette. Clicking on one of the palette's buttons places the corresponding character/expression at the current input location. This particular palette probably appears by default when you start *Mathematica*, but if not, you can access it or any of the other palettes through your Palette menu.



■ The Documentation Center

Mathematica's Documentation Center may be accessed by selecting Documentation Center from the Help menu. Among the wealth of information available through the Documentation Center are descriptions of all of *Mathematica*'s built-in functions, including examples of their use and links to related tutorials. You can also enter commands from within the Documentation Center. (Whatever you change there will not be saved.)



The Documentation Center provides a great deal of tutorial material. If you're a beginner, we suggest that you peruse the tutorials found by entering each of the following in the Documentation Center search field.

tutorial/UsingTheMathematicaSystemOverview

tutorial/InputAndOutputInNotebooksOverview

tutorial/BuildingUpCalculationsOverview

tutorial/EnteringTwoDimensionalExpressionsOverview

tutorial/GraphicsAndSoundOverview

(Hopefully future updates to the Documentation Center will provide a more convenient way to access these and other tutorials.)

■ Basic Calculations

Mathematica allows multiplication to be indicated in three ways. Expressions separated by a space are multiplied. (Note that the system automatically replaces the space with ×.)

```
217 × 5713
```

```
1 239 721
```

An asterisk between expressions indicates multiplication.

```
321 * 5.479
```

```
1758.76
```

When there is no ambiguity, juxtaposed expressions (without spaces) are understood and multiplied.

```
2x
```

```
2 x
```

More than one command can be given in one input cell. A single input cell may consist of two or more lines. A new line within the current cells is obtained by pressing "return." Commands on the same line within a cell must be separated by semicolons. *The output of any command that ends with a semicolon is not displayed*.

```
a = 17 / 13 + 211 / 93;
b = 23 a; c = (a + b) / 51
```

$$\frac{34\,592}{20\,553}$$

The percent sign % refers to the last output (not necessarily the preceding cell).

```
3 / 17 + 1 / 5
```

$$\frac{32}{85}$$

```
%^2
```

$$\frac{1024}{7225}$$

If the last command on any line is not followed by a semicolon, its result *is* displayed. This effect is very handy for showing intermediate steps in a calculation. The following computes $\frac{25!}{3!\times 22!}(.1)^3(.9)^{22}$ (a *binomial probability*).

```
25 !
% / (3 ! × 22 !)
% * .1^3 .9^22
```

```
15 511 210 043 330 985 984 000 000
```

```
2300
```

```
0.226497
```

You should avoid use of the percent sign as much as possible—especially in separate cells. It is *far* better to give names to results and to use those names in subsequent calculations.

■ Parentheses, Brackets, and Braces

The syntax of *Mathematica* is absolutely strict and consistent (and quite simple once you get used to it). For that reason, there are some differences between *Mathematica*'s syntax and the often inconsistent and sometimes ambiguous mathematical notation that we're all used to. For example:

Parentheses are used *only* for grouping expressions.

```
x (x + 2)^2
```

```
x (2 + x)^2
```

Brackets are used *only* to enclose the argument(s) of a function.

```
Cos[π / 3]
```

$$\frac{1}{2}$$

Braces are used *only* to enclosed the elements of a *list* (which might represent a set, an ordered pair, or even a matrix).

```
{1, 2, 3, 4}
```

```
{1, 2, 3, 4}
```

Consequently, *Mathematica* does *not* understand what you intend by entering any of these expressions, for example:

```
[x + y (1 - y)]^2
```

```
Syntax::tsntxi : "[x + y (1 – y)]" is incomplete; more input is needed.
Syntax::sntxi : Incomplete expression; more input is needed.
```

```
(1, 2)
```

```
Syntax::sntxf : "(" cannot be followed by "1, 2)".
Syntax::tsntxi : "1, 2" is incomplete; more input is needed.
```

```
Sin (π)
```

```
π Sin
```

In these first two instances, we were lucky to get an error message. But in the last one, *Mathematica* simply multiplied the expressions π and `Sin`—with no complaint at all!

■ Symbolic vs. Numerical Computation

Computations are typically done symbolically (and therefore *exactly*), unless we request otherwise.

```
123 / √768
```

$$\frac{41\sqrt{3}}{16}$$

One way to obtain a numerical result is to use the numerical evaluation function, `N`.

```
N[123 / Sqrt[768]]
```

```
4.43838
```

We also get a numerical result if any of the numbers in the expression are made numerical by use of a decimal point.

```
123. / Sqrt[768]
```

```
4.43838
```

Unless we *cause* a numerical result, *Mathematica* typically returns an *exact* form, which in many cases is identical to the expression entered.

```
Cos[π / 12]
```

$$\frac{1+\sqrt{3}}{2\sqrt{2}}$$

```
Log[2]
```

```
Log[2]
```

■ Names and Capitalization; Basic Functions

All built-in *Mathematica* objects—functions, constants, options, etc.—have full names that begin with a capital letter (or in the case of certain "global" parameters, a dollar sign followed by a capital letter).

```
Sin[π / 3]
```

$$\frac{\sqrt{3}}{2}$$

```
PrimeQ[22 801 763 489]
```

```
True
```

```
Solve[x^2 + x - 12 == 0, x]
```

```
{{x → -4}, {x → 3}}
```

These full names are used internally by *Mathematica*, even when it is far more natural for us to use a symbolic form such as $x + 7$. `FullForm` lets us see the internal representation of an expression.

```
FullForm[x + 7]
```

```
Plus[7, x]
```

```
FullForm[x == x^2]
```

```
Equal[x, Power[x, 2]]
```

When the name of a built-in *Mathematica* object is made of two or more words, all of the component words are capitalized. Some typical *Mathematica*-style names are `FindRoot`, `PlotRange`, `AspectRatio`, `NestList`, etc. In almost all cases the component words are spelled out in full.

All of the familiar "elementary functions" are built-in. In some cases—if you remember to capitalize the first letter and to use brackets instead of braces—you would guess correctly how to use one of those functions. For example,

```
Sin[π / 12]
```

$$\frac{-1+\sqrt{3}}{2\sqrt{2}}$$

There are a few things in this regard that should be pointed out. First, the inverse trigonometric functions use the "arc-function" convention:

```
ArcTan[1]
```

$$\frac{\pi}{4}$$

```
ArcCos[1 / 2]
```

$$\frac{\pi}{3}$$

💡 Also, the natural logarithm is `Log`, not `Ln`.

```
Log[E]
```

```
1
```

■ Algebraic Manipulation

Mathematica is an example of a type of software system that is often called a *computer algebra system*. In addition to numerical computations, a computer algebra system also does *symbolic computation* including the manipulation of algebraic expressions. *Mathematica* has a number of functions for this purpose. Among these are `Expand`, `Factor`, `Together`, and `Apart`.

```
Expand[(x + 5)^3 (2 x - 1)^2]
```

```
125 - 425 x + 215 x^2 + 241 x^3 + 56 x^4 + 4 x^5
```

```
Factor[x^3 + 2 x^2 - 5 x - 6]
```

```
(-2 + x) (1 + x) (3 + x)
```

```
Together[x + 2/(x^2 + 1)]
```

$$\frac{2+x+x^3}{1+x^2}$$

```
Apart[x/(x^2 + 3 x + 2)]
```

$$-\frac{1}{1+x}+\frac{2}{2+x}$$

Notice that *Mathematica* does not automatically simplify algebraic expressions:

```
x (3 - x) - 5 x^2 + (x - 1) (2 x + 3)
```

```
(3 - x) x - 5 x^2 + (-1 + x) (3 + 2 x)
```

`Simplify` can be used for this purpose.

```
Simplify[x (3 - x) - 5 x^2 + (x - 1) (2 x + 3)]
```

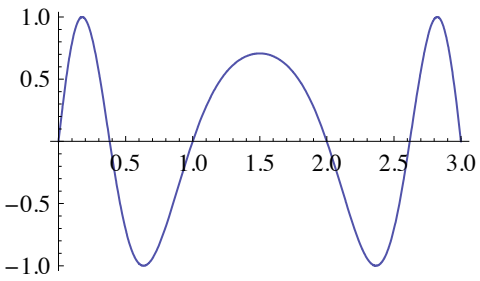
```
-3 + 4 x - 4 x^2
```

■ Plotting Graphs: An Introduction to Options

Mathematica is extremely good at creating graphics to help us analyze problems. We will be primarily interested in graphing functions of one variable. This is done with `Plot`.

The function $f(x) = \sin(\pi x (3 - x))$ is graphed on the interval $0 \le x \le 3$ as follows.

```
Plot[ Sin[π x (3 - x)], {x, 0, 3}]
```

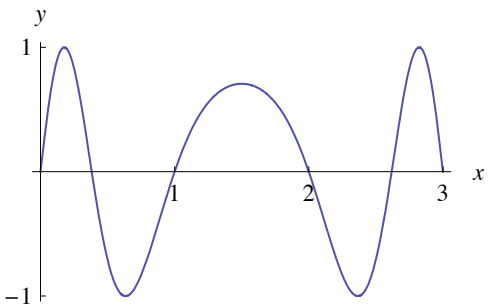


Notice that two *arguments* are provided to `Plot`. The first is our function in the form of an *expression*, and the second is a *list* with three members, specifying (i) the name of the variable, (ii-iii) the left and right endpoints of the interval.

There are numerous ways that we could have affected the appearance of the plot by specifying **options**. Among the options for `Plot` are `PlotRange`, `Ticks`, `AxesLabel`, `AspectRatio`, and `PlotStyle`.

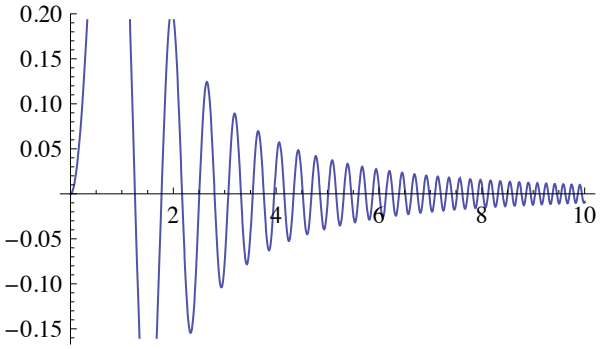
The following creates a plot with labelled axes with no tick marks. Note that the arrow character is typed as ESC->ESC. (Actually, just -> will do.)

```
Plot[Sin[π x (3 - x)], {x, 0, 3},
 Ticks → {{1, 2, 3}, {-1, 1}}, AxesLabel → {x, y}]
```



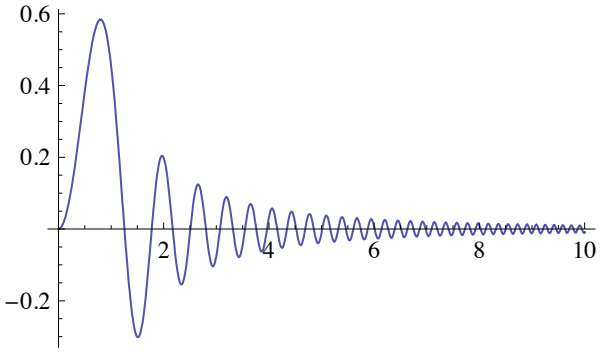
Notice that the following plot chops off high and low parts of the curve.

```
Plot[Sin[2 x^2]/(x^2 + 1), {x, 0, 10}]
```



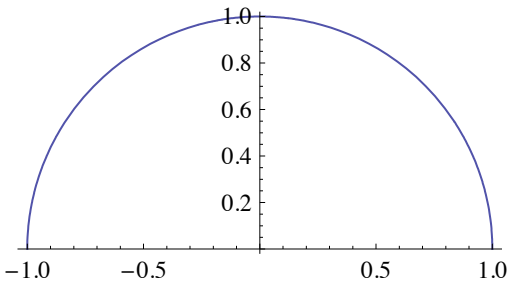
This can be cured with the `PlotRange` option.

```
Plot[Sin[2 x^2]/(x^2 + 1), {x, 0, 10}, PlotRange → All]
```



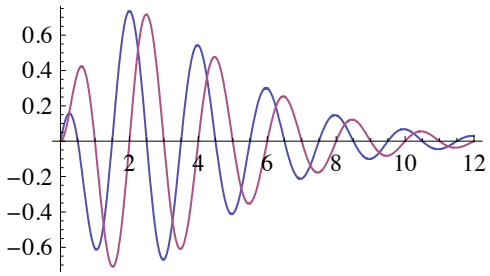
Without our specifying `AspectRatio → Automatic`, the following semicircle would be stretched vertically.

```
Plot[Sqrt[1 - x^2], {x, -1, 1}, AspectRatio → Automatic]
```



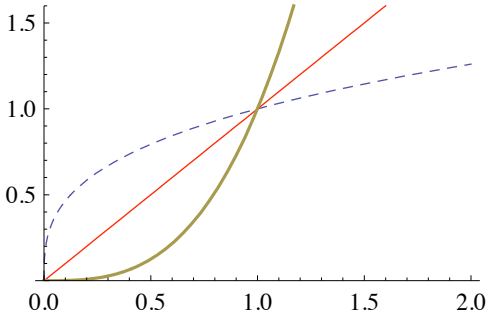
To plot more than one function at once, we give `Plot` a *list* of functions.

```
Plot[{x e^(-x/2) Cos[π x], x e^(-x/2) Sin[π x]}, {x, 0, 12}]
```



When plotting multiple functions, it is often desirable to plot them with different styles. The `PlotStyle` option lets us do that.

```
Plot[{∛x, x, x^3}, {x, 0, 2}, PlotRange → {0, 1.6},
 PlotStyle → {{Dashing[{.02}]}, {Red}, {Thickness[.007]}}]
```



■ Variables

As we mentioned earlier, all built-in *Mathematica* objects begin with an upper-case letter. For that reason, it is usually a good idea to use variable names that begin with a lower-case letter. In this manual we will loosely follow that convention. It is also good practice to give meaningful names to variables and never to make assignments to single letter variables.

Assignments to variables are remembered by *Mathematica* (for the duration of one "kernel session") until the variable is "cleared." *This is probably the single most important thing to remember when you run into difficulties using Mathematica*. (We will have more to say on this in Section 1.8.)

```
piSixths = N[π / 6]
```

```
0.523599
```

```
Clear[piSixths]; piSixths
```

```
0.523599
```

```
piSixths
```

Assignments in *Mathematica* are made in two ways: (i) with a single equal sign, or (ii) with a colon followed by an equal sign. For simple assignments such as

```
radius := √(10. / π)
```

it makes little difference which method is used. In this particular case, the consequence of using := is that `radius` has not yet been computed. (Also notice that no output is produced when := is used.) The evaluation has been delayed until we cause it to be done—for example, by entering

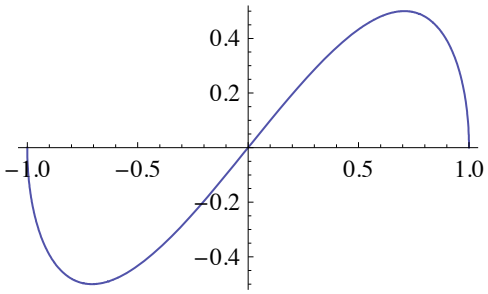
```
radius
```

```
1.78412
```

A better example to illustrate delayed evaluation with := is as follows. If we assign a plot to a variable with :=, then no plot is created. The variable is assigned *the command itself*, not the result.

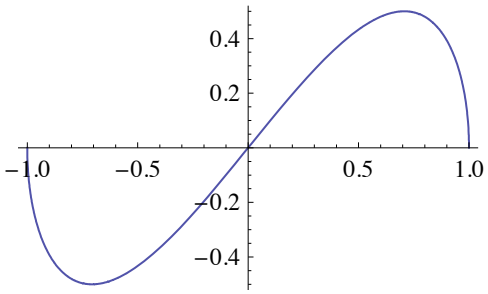
```
graph := Plot[x √(1 - x^2) , {x, -1, 1}]
```

```
graph
```



If we assign a plot to a variable with =, then the plot is created and the variable is assigned the resulting *graphics object*.

```
graph = Plot[x √(1 - x^2) , {x, -1, 1}]
```



We will address these issues again in Section 1.2. (See also Exercise 9 in this section.)

■ Tips and Shortcuts

We end this quick tour of *Mathematica* with a few tips and shortcuts with respect to typing expressions.

■ Entering Exponents, Radicals, and Fractions

To enter an exponent or superscript, press CTRL-[6]. (Note that this is analogous to the SHIFT-[6] caret (^), which is used for exponentiation in `InputForm`.) To leave the resulting exponent "box," press [→] or CTRL-SPACE.

To enter a subscript, press CTRL-[-]. (This is analogous to the SHIFT-[-] underscore character (_), which is used to create subscripts in the TEX typesetting language.) To leave the resulting subscript "box," press [→] or CTRL-SPACE.

To enter an expression involving a square root, press CTRL-[2]. To leave the resulting square root box, press [→] or CTRL-SPACE.

To enter a fraction, press CTRL-[/]. To move from the numerator box to the denominator box, press TAB To exit the fraction, press [→] or CTRL-SPACE.

■ Greek Letters and Other Special Characters

Many special characters and symbols can be typed easily by pressing the ESC key before and after typing some easily remembered standard character(s). For instance, to type the Greek letter α (alpha), just type ESC `a` ESC or ESC `alpha` ESC. Other Greek letters can be typed similarly.

Other shortcuts for common special characters include:

ESC `ee` ESC produces the constant e, the base of the natural logarithm.

ESC `ii` ESC produces the imaginary number $i = \sqrt{-1}$.

ESC `int` ESC produces an integral sign $\int$.

ESC `pd` ESC produces a derivative operator ∂.

ESC `inf` ESC produces an infinity symbol ∞.

Of course, you may prefer to use the buttons on the BasicInput palette.

We should also point out that the familiar numbers denoted by e, i, and π can be entered as such or as `E`, `I`, and `Pi`, respectively.

```
{e == E, i == I, π == Pi}
```

```
{True, True, True}
```

Be careful to notice, however, that an ordinary e is not the same as e, and an ordinary i is not the same as i.

◆ Exercises

1. Compute both an exact and a numerical value for each of the following numbers.

   a) $\left(23^3 - 3\,(117-48)^2\right) \Big/ \sqrt{7^5 - 5^7}$ b) $\cos\frac{319\,\pi}{12}$ c) $\frac{83!}{111!}$ d) $\ln 2981$

2. Use `Simplify` on each of the following.

   a) $\ln\left(2e^5\right)$ b) $1 + \cos 2x$ c) $\dfrac{x + (x\,(x-1))^3 - 4}{x^2 + x - 6}$

3. Factor each of these polynomials:

   a) $6x^3 + 47x^2 + 71x - 70$ b) $12x^6 - 56x^5 + 100x^4 - 80x^3 + 20x^2 + 8x - 4$

4. Plot the function $f(x) = \frac{\sin(x^3)}{x^3}$ on the interval $0 \le x \le 1$ with:

   a) no options b) `PlotRange→All`

   c) `PlotRange→All, AspectRatio→Automatic`

5. Create a plot containing the graphs of $y = x^2$ and $y = x^5$ over $0 \le x \le 2$ with:

   a) no options b) `PlotRange→All` c) `PlotRange→{0,2}`

   d) `PlotStyle→{Red,Blue}` e) `PlotStyle→{{Red,Thick},{Blue,Thick}}`

6. Look up each of the following functions in the Documentation Center, and then plot them on the indicated interval.

   a) `Floor`, $0 \le x \le 10$ b) `PrimePi`, $0 \le x \le 100$

7. Use the Documentation Center to learn what `RandomReal[]` and `RandomInteger[]` do. Then enter and the following and explain the output:

```
GraphicsRow[{Plot[RandomReal[] x, {x, 0, 1}],
  Plot[RandomInteger[] x, {x, 0, 1}]}]
```

8. `RandomReal` and `RandomInteger` provide a good illustrations of the difference between using `=` and using `:=` in an assignment. Enter each of the following several times. Describe and explain the difference in the results.

```
r = RandomInteger[100]; Table[r, {10}]
```

   and

```
r := RandomInteger[100]; Table[r, {10}]
```

1.2 Functions

■ Defining Functions

■ Blank

When defining a function, it is essential to follow each argument by a `Blank` (or "underscore"). Also, recall that the arguments of a function are enclosed by brackets. For example, we would define the function $f(x) = x^3 - 2$ by entering

```
f[x_] := x^3 - 2 x
```

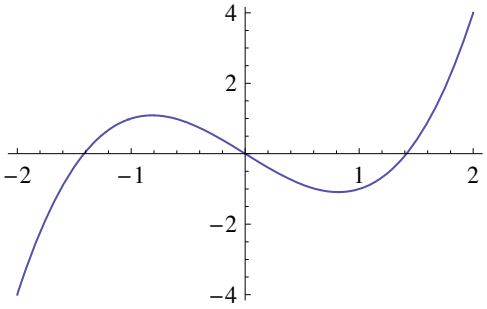
Then we can evaluate the function at any number

```
f[3]
```

```
21
```

or plot its graph:

```
Plot[f[x], {x, -2, 2}]
```



⚠ Note that the `Blank` appears next to `x` *only on the left side of the expression*. Also, when defining a function with more than one argument, a `Blank` must follow each one. For instance:

```
g[x_, y_, z_] := x + y + z
g[1, 2, 3]
```

```
6
```

■ Set (=) versus SetDelayed (:=)

Definitions of functions—and assignments of expressions to variables in general—can be made using either "equal" or "colon-equal." The difference between these two ways is described by the full *Mathematica* names of the `=` and `:=` symbols, which are `Set` and `SetDelayed`.

When an assignment is made using `Set`, any calculations that are indicated on the right side are done as the assignment is entered. When an assignment is made using `SetDelayed`, any calculations that are indicated on the right side are delayed until the defined expression is used.

In many cases, such as in the definitions of `f` and `g` above, it makes no difference which is used. To see a simple example that indicates the importance of using `Set` rather than `SetDelayed`, let's suppose we want to define $f(x)$ to be the derivative of $(x + 1)\cos x$. If we enter

```
f[x_] := ∂x ((x + 1) Cos[x])
```

notice what happens when we try to evaluate $f(2)$:

```
f[2]
```

```
General::ivar : 2 is not a valid variable. ≫
```

```
∂2 (3 Cos[2])
```

However, if we enter

```
f[x_] = ∂x ((x + 1) Cos[x])
```

```
Cos[x] - (1 + x) Sin[x]
```

then f works the way we want it to:

```
f[2]
```

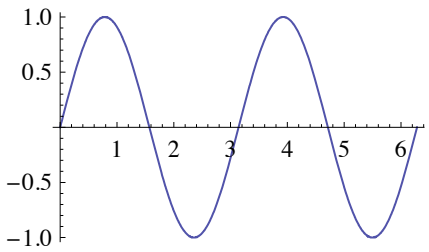
```
Cos[2] - 3 Sin[2]
```

When is it important to use `SetDelayed` rather than `Set`? Here's a simple example: Suppose that we want to plot the graph of $\sin kx$ for several values of k and that we decide to define a function as follows to create each of the desired plots.

```
plotSin[k_] := Plot[Sin[k x], {x, 0, 2 π}]
```

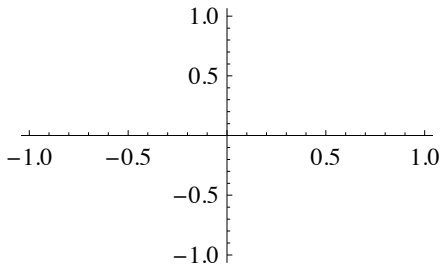
For instance, with $k = 2$, we get the following graph:

```
plotSin[2]
```



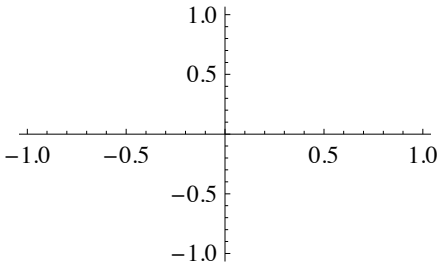
But what would have happened if we used `=` rather than `:=` in the definition of `plotSin`? *Mathematica* would attempted to plot the graph of $\sin kx$ immediately when the definition is entered. This doesn't work because k has no numerical value, and an empty plot results.

```
plotSin[k_] = Plot[Sin[k x], {x, 0, 2 π}, PlotRange → All]
```



The empty plot that was produced and assigned to `plotSin[k_]` will now be the "value" of `plotsin[k]` for any k; for instance:

```
plotSin[2]
```



■ Applying Functions with @ and //

Suppose that we define a simple function such as

```
f[x_] := x (x - 1)
```

Naturally, we could evaluate this function at, say, $x = 3$, by entering

```
f[3]
```

```
6
```

There are two other ways to do the same thing. We can evaluate *f at* 3 by entering

```
f@3
```

```
-1
```

On the other hand, we can apply *f* to 3 like this:

```
3 // f
```

```
6
```

We will use this *postfix* method of function application frequently, often for the purpose of applying either `Simplify` to some expression or `N` to some numerical calculation. For example, we will use the following style when doing a symbolic calculation:

```
2 x + x (5 x + 1) // Simplify
```

```
x (3 + 5 x)
```

When doing an exact numerical calculation, we will commonly use a style that is similar but displays the exact value followed by the numerical value:

```
√585 / 33
```

```
% // N
```

$$\frac{\sqrt{65}}{11}$$

```
0.732933
```

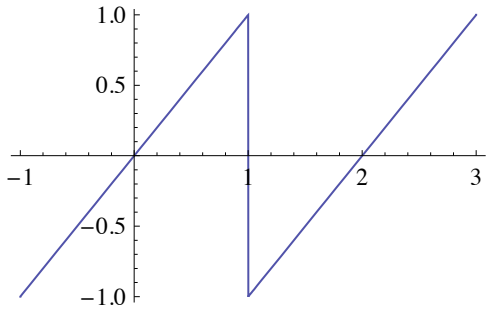
■ Piecewise-defined Functions

Mathematica has two primary logical functions that we can use to enter definitions of piecewise- defined functions. These are `If` and `Which`. `If` usually works best for functions with two pieces, such as

$$f(x) = \begin{cases} x, & \text{if } x \le 1; \\ x - 2, & \text{if } x > 1. \end{cases}$$

This function can be entered and plotted as follows.

```
f[x_] := If[x ≤ 1, x, x - 2]
Plot[f[x], {x, -1, 3}]
```

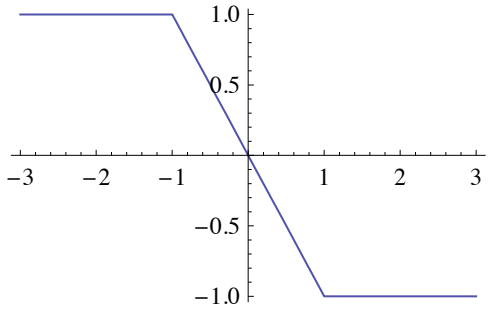


More complicated functions are better handled with `Which`. For example, the function

$$f(x) = \begin{cases} 1, & \text{if } x \le -1 \\ -x, & \text{if } -1 < x \le 1 \\ -1, & \text{if } x > 1 \end{cases}$$

can be entered and plotted as follows.

```
f[x_] := Which[x ≤ -1, 1, -1 < x ≤ 1, -x, x > 1, -1]
Plot[f[x], {x, -3, 3}]
```



■ Piecewise

`Piecewise` was new in *Mathematica* 5.1 and is similar to `Which`. While `Which` must be supplied with an alternating sequence of conditions and values, *i.e.*,

`Which`[$cond_1, val_1, cond_2, val_2, \ldots$],

`Piecewise` takes a single list containing value-condition *pairs*, as in

$$\texttt{Piecewise[}\{\{val_1, cond_1\}, \{val_2, cond_2\}, \ldots\}\texttt{]}$$

For example, the function

$$f(x) = \begin{cases} 1, & \text{if } x \le -1 \\ -x, & \text{if } -1 < x \le 1 \\ -1, & \text{if } x > 1 \end{cases}$$

can be input as follows. Notice the automatic display of the output in "left-bracket form."

```
f[x_] = Piecewise[{{1, x ≤ -1}, {-x, -1 < x ≤ 1}, {-1, 1 < x}}]
```

```
⎧ 1    x ≤ -1
⎨ -x   -1 < x ≤ 1
⎩ -1   1 < x
```

It is also possible to enter such a definition in the same left-bracket form as follows: Press ESC pw ESC followed by CTRL RET. Then you'll have a template like this:

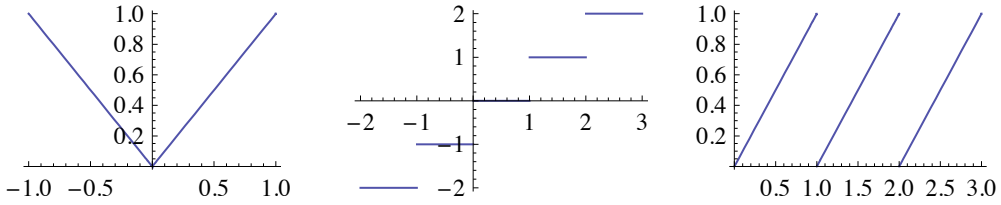
```
⎧ □  □
⎩ □  □
```

Pressing CTRL RET again will add another row.

■ `Abs`, `Floor`, and `Mod`

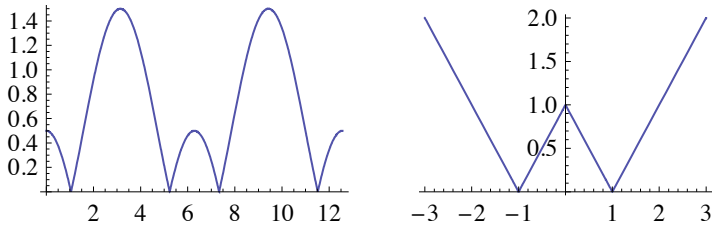
There are a handful of built-in piecewise-defined functions, including `Abs`, `Floor`, and `Mod`, whose graphs are shown below.

```
GraphicsRow[
 {Plot[Abs[x], {x, -1, 1}], Plot[Floor[x], {x, -2, 3}],
Plot[Mod[x, 1], {x, 0, 3}]}]
```



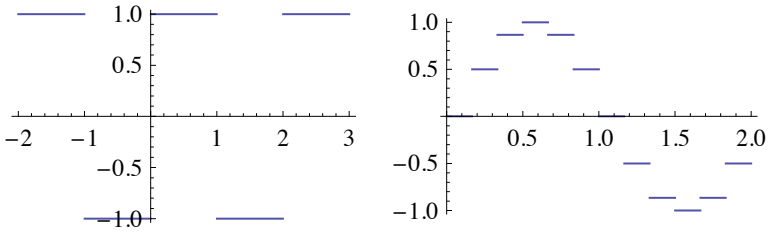
The following functions are built with `Abs`.

```
GraphicsRow[{Plot[Abs[Cos[x] - .5], {x, 0, 4 π}],
  Plot[Abs[Abs[x] - 1], {x, -3, 3}]}]
```



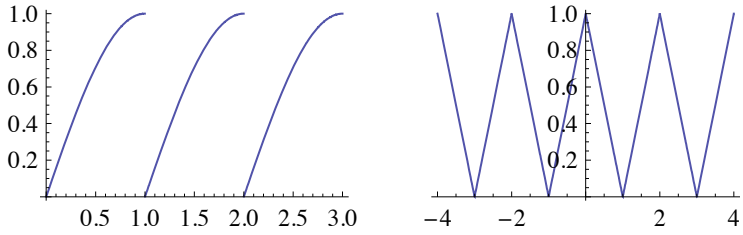
`Floor` helped create each of these *step functions*:

```
GraphicsRow[{Plot[(-1)^Floor[x], {x, -2, 3}],
  Plot[Sin[π Floor[6 x] / 6], {x, 0, 2}]}]
```



A *periodic function* based on a piece of any graph can be constructed with `Mod`.

```
GraphicsRow[{Plot[Sin[π Mod[x, 1] / 2], {x, 0, 3}],
  Plot[Abs[Mod[x, 2] - 1], {x, -4, 4}]}]
```



◆ Exercises

1. Enter definitions for each of $f(x) = x(x-2)^2$, $g(x) = x - 3$, and $h(x) = x^2 - 1$ and then compute and simplify each of the following:

   a) $f(g(h(x)))$ b) $h(g(f(x)))$ c) $f(h(g(x)))$

2. Enter a definition for $f(x) = \left(1 + x^2\right)^{-1}$. Then create a plot (over $-3 \le x \le 3$) showing the graphs of:

   a) $f(x)$, $f(x-1)$, and $f(x+1)$ b) $f(x)$, $f(2x)$, and $f(8x)$

3. Using `If`, define and plot each of a) $f(x) = \begin{cases} -x, & \text{if } x < 0 \\ x^2 - 1, & \text{if } x \ge 0 \end{cases}$ b) $f(x) = \begin{cases} 1, & \text{if } x \le \pi/2 \\ \sin x, & \text{if } x > \pi/2 \end{cases}$

4. Using `Which`, define and plot each of

   a) $f(x) = \begin{cases} 1, & \text{if } x < 0 \\ 1 - x^2, & \text{if } 0 \le x \le 2 \\ -3, & \text{if } x > 2 \end{cases}$ b) $f(x) = \begin{cases} x + 1, & \text{if } x < 0 \\ 1 - 2x, & \text{if } 0 \le x \le 1 \\ x - 2, & \text{if } x > 1 \end{cases}$

5. Redo Exercise 4 with `Piecewise`.

6. Plot each of the functions:

   a) $f(x) = (x - \text{Floor}(x))^2$ b) $f(x) = x\,(-1)^{\text{Floor}(x)}$

7. The function `Mod` provides an easy way to create a *periodic* function from a simpler function that describes a single period. For example, the function

```
f0[x_] := Sqrt[1 - (x - 1)^2]
```

   describes the top half of the circle with radius 1 centered at (1, 0). Plot its graph by entering

```
Plot[f0[x], {x, 0, 2},
 AspectRatio → Automatic, PlotRange → {-.5, 1.5}]
```

   The corresponding periodic function with period 2 is

```
f[x_] := f0[Mod[x, 2]]
```

   In a similar manner, define and plot (for $0 \le x \le 9$) a periodic function with period 3 that agrees with $f_0(x) = e^{-x}$ on the interval $0 \le x < 3$.

8. The *unit step function* is defined by $u(x) = \begin{cases} 0, & x < 0, \\ 1, & x \ge 0. \end{cases}$ Its *Mathematica* name is `UnitStep`.

   a) Using `UnitStep`, plot the graph of u on $-1 \le x \le 2$.

   b) Plot $f(x) = u(x - 1)$ on $-1 \le x \le 2$ and express $f(x)$ in piecewise form.

   c) Plot $f(x) = u(x - 1)\sin(2\pi x)$ on $-1 \le x \le 2$ and express $f(x)$ in piecewise form.

   d) Plot $f(x) = u(x - 2) - h(x - 1)$ on $0 \le x \le 3$ and express $f(x)$ in piecewise form.

   e) Plot $f(x) = (u(x - 2) - u(x - 1))\sin(2\pi x)$ on $0 \le x \le 3$ and express $f(x)$ in piecewise form.

1.3 Equations

Certainly one of the most frequent mathematical tasks that we need to do is to solve an equation. In order to be able to solve equations with *Mathematica*, we first need to understand how equations are formed. The important thing to remember is that *double equal signs* are used to form an equation.

Actually, double equal signs constitute a *logical test* that returns either `True`, `False`, or the equation itself.

```
2 * 17 - 34 == 0
```

```
True
```

```
3 == 4
```

```
False
```

```
x^2 - 4 == 0
```

```
-4 + x^2 == 0
```

Notice, however, that *Mathematica* returns `True` only when the expressions on each side are identical. Only the most superficial simplification is done prior to the test, as in:

```
2 x + x - 2 == 3 x + 5 - 7
```

```
True
```

Notice that for this equation:

```
x^2 - 4 == (x + 2) (x - 2)
```

$$-4 + x^2 == (-2 + x)\ (2 + x)$$

a nontrivial operation must be done to one side or the other before the expressions become truly identical. Finally, notice that *Mathematica* returns an error message if we use a single equal sign improperly.

```
3 = 4
```

```
Set::setraw : Cannot assign to raw object 3. ≫
```

```
4
```

The single equal sign is used only for *assignments* such as

```
area = π r^2
```

$$\pi\, r^2$$

Mathematica has three functions for solving ordinary equations. These are `Solve`, `NSolve`, and `FindRoot`. `Solve` works very well on polynomial and many other algebraic equations.

```
Solve[6 x^3 - 23 x^2 + 25 x - 6 == 0, x]
```

$$\left\{\left\{x \to \frac{1}{3}\right\}, \left\{x \to \frac{3}{2}\right\}, \{x \to 2\}\right\}$$

```
Solve[x^4 - 2 x^3 - x^2 + 6 x - 6 == 0, x]
```

$$\left\{\{x \to 1 - i\}, \{x \to 1 + i\}, \left\{x \to -\sqrt{3}\right\}, \left\{x \to \sqrt{3}\right\}\right\}$$

```
Solve[Sqrt[x - 1] + x == 4 + Sqrt[x + 4], x]
```

$$\{\{x \to 5\}\}$$

`Solve` will also give solutions to trigonometric (or exponential/logarithmic) equations, but frequently gives a warning.

```
Solve[2 Sin[2 x] == Cos[x]^2 - 1, x]
```

```
Solve::ifun :
  Inverse functions are being used by Solve, so some solutions may not be found;
    use Reduce for complete solution information. ≫
```

$$\left\{\{x \to 0\}, \{x \to -\pi\}, \{x \to \pi\}, \left\{x \to \text{ArcCos}\left[-\frac{1}{\sqrt{17}}\right]\right\}, \left\{x \to -\text{ArcCos}\left[\frac{1}{\sqrt{17}}\right]\right\}\right\}$$

`Solve` will also find solutions of a system of equations. The equations must be given as elements of a list, *i.e.*, separated by commas and enclosed in braces.

```
Solve[{x^2 - y == 1, -x + y == 1}, {x, y}]
```

```
{{y → 0, x → -1}, {y → 3, x → 2}}
```

The solutions of polynomial equations of degree five or greater generally cannot be found in any exact form. Notice how *Mathematica* "avoids" the problem:

```
Solve[x^5 - 10 x^2 + 5 x + 1 == 0, x]
```

```
{{x → Root[1 + 5 #1 - 10 #1^2 + #1^5 &, 1]},
 {x → Root[1 + 5 #1 - 10 #1^2 + #1^5 &, 2]},
 {x → Root[1 + 5 #1 - 10 #1^2 + #1^5 &, 3]},
 {x → Root[1 + 5 #1 - 10 #1^2 + #1^5 &, 4]},
 {x → Root[1 + 5 #1 - 10 #1^2 + #1^5 &, 5]}}
```

In such situations, we can always resort to numerical solutions. `NSolve` finds a numerical approximation to each solution of a polynomial equation, including complex solutions.

```
NSolve[x^5 - 10 x^2 + 5 x + 1 == 0, x]
```

```
{{x → -1.22065 - 1.89169 i}, {x → -1.22065 + 1.89169 i},
 {x → -0.153102}, {x → 0.66946}, {x → 1.92494}}
```

In many situations where `Solve` *is* successful, such as:

```
Solve[x^3 - 10 x^2 + 5 x + 1 == 0, x]
```

$$\left\{\left\{x \to \frac{10}{3} + \frac{85}{3\left(\frac{1}{2}\left(1523+9\,i\sqrt{1691}\right)\right)^{1/3}} + \frac{1}{3}\left(\frac{1}{2}\left(1523+9\,i\sqrt{1691}\right)\right)^{1/3}\right\},\right.$$
$$\left\{x \to \frac{10}{3} - \frac{1}{6}\left(1+i\sqrt{3}\right)\left(\frac{1}{2}\left(1523+9\,i\sqrt{1691}\right)\right)^{1/3} - \frac{85\left(1-i\sqrt{3}\right)}{3\,2^{2/3}\left(1523+9\,i\sqrt{1691}\right)^{1/3}}\right\},$$
$$\left.\left\{x \to \frac{10}{3} - \frac{1}{6}\left(1-i\sqrt{3}\right)\left(\frac{1}{2}\left(1523+9\,i\sqrt{1691}\right)\right)^{1/3} - \frac{85\left(1+i\sqrt{3}\right)}{3\,2^{2/3}\left(1523+9\,i\sqrt{1691}\right)^{1/3}}\right\}\right\}$$

it may still be preferable to use `NSolve`:

```
NSolve[x^3 - 10 x^2 + 5 x + 1 == 0, x]
```

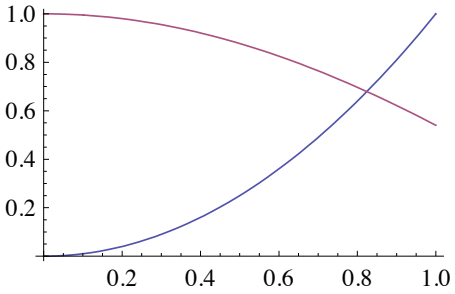
```
{{x → -0.152671}, {x → 0.692369}, {x → 9.4603}}
```

Many equations *require* the use of `FindRoot`, which incorporates a numerical procedure. For example, consider

$$x^2 = \cos x.$$

`FindRoot` can find only one solution at a time and requires us to supply an initial guess at the solution we're looking for. An appropriate initial guess can usually be determined by examining a graph.

```
Plot[{x^2, Cos[x]}, {x, 0, 1}]
```




```
FindRoot[x^2 == Cos[x], {x, .85}]
```

```
{x → 0.824132}
```

◆ Exercises

1. The equation $x^3 = x + 1$ has one real solution. Find its exact value with `Solve` and its numerical value with `NSolve`.

2. Use `Solve` to find the solution(s) of each of the systems:

   a) $2x + 3y = 1,\ x^2 + y^2 = 1$ b) $3x - 2y = 5,\ 7x + 3y = 2$

   c) $x + y + z = 2,\ x - y + z = 1,\ x^2 + y^2 + z^2 = 2$

3. Use `Solve` on the *underdetermined* system

   $$x + y + 2z = 2,\ \ x - 2y + z = 1$$

   and interpret the result. Try each combination of *solve variables*: `{x,y,z}`, `{x,y}`, `{x,z}`, and `{y,z}`. Which gives the "cleanest" solution?

4. For each of the following functions, plot the graph to determine the approximate location of each of its zeros. Then find each of the zeros with `FindRoot`.

   a) $f(x) = e^{-x/2}$ b) $f(x) = x - 9\cos x$ c) $f(x) = x^2 - \tan^{-1} x$

5. For each of the following equations, plot both sides of the equation to determine the approximate location of each of its solutions in the specified interval. Then find each of the solutions with `FindRoot`.

   a) $\sin x \cos 2x = \cos x \sin 3x,\ 0 \le x \le 2\pi$ b) $\sin x^2 = \sin^2 x,\ 0 \le x \le \pi$

   c) $\tan x = x,\ 0 \le x \le 3\pi$ d) $\sin x = e^{-x},\ \ 0 \le x \le 4$

6. Two spheres have a combined volume of 148 cubic inches and a combined surface area of 160 square inches. Find the radii of the two spheres.

7. Two spheres have a combined volume of 148 cubic inches and a combined surface area of 160 square inches. Find the radii of the two spheres.

8. An open-topped aquarium holds 40 cubic feet of water and is made of 60 square feet of glass. The length of the aquarium's base is twice its width. Find the dimensions of the aquarium.

9. a) Find the equation of the parabola that passes through the points $(-1, 1)$, $(1, 2)$, and $(2, 3)$.

   b) Find the cubic polynomial $f(x)$ such that $f(1) = f(2) = f(3) = 1$ and $f(4) = 7$.

1.4 Lists

Lists are ubiquitous in *Mathematica*. A list is anything that takes the form of a series of objects separated by commas and enclosed in braces, such as:

```
{a, b, c}
```

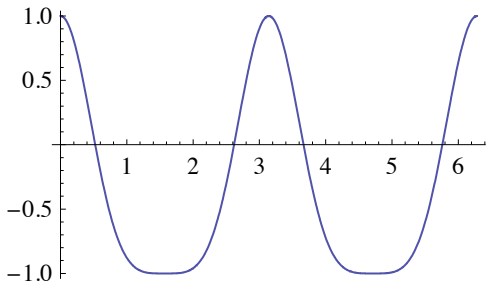
```
{{1, 3}, {2, 5}}
```

```
{x, 1, 2}
```

```
{x^2 + y == 2, 2 x - y == 0}
```

Many built-in commands expect lists for certain arguments. For example, in

```
Plot[Cos[π Sin[x]], {x, 0, 2 π}]
```



the second argument is a list that specifies the variable name and the interval over which to plot. In

```
Solve[{x^2 + y == 2, 2 x - y == 0}, {x, y}]
```

$$\{\{y \to 2\,(-1-\sqrt{3}),\ x \to -1-\sqrt{3}\},\ \{y \to 2\,(-1+\sqrt{3}),\ x \to -1+\sqrt{3}\}\}$$

each of the two arguments is a list, and the result is also a list (of lists).

■ Listable Functions

Most built-in *Mathematica* functions and operations are *listable*. When a listable function is applied to a list, it is applied to each element of the list and returns the result as a list. For example,

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}^2
```

```
{1, 4, 9, 16, 25, 36, 49, 64, 81}
```

```
Sqrt[{{{2, 3}, {4, 5}}, {{6, 7}, {8, 9}}}]
```

$$\{\{\{\sqrt{2},\sqrt{3}\},\{2,\sqrt{5}\}\},\{\{\sqrt{6},\sqrt{7}\},\{2\sqrt{2},3\}\}\}$$

```
1/{1, 2, 3, 4, 5}
```

$$\left\{1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\frac{1}{5}\right\}$$

```
{1, 2, 3} + {4, 5, 6}
```

```
{5, 7, 9}
```

```
{1, 2, 3} {4, 5, 6}
```

```
{4, 10, 18}
```

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

```
{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}
```

```
Cos[{0, π / 4, π / 2, 3 π / 4, π}]
```

$$\left\{1, \frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}}, -1\right\}$$

■ The Parts of a List

Here's a simple list:

```
alist = {2, x, y, {a, b}}
```

```
{2, x, y, {a, b}}
```

Notice that it has four parts.

```
Length[alist]
```

```
4
```

Here's the third part:

```
alist[[3]]
```

```
y
```

The fourth part is itself a list:

```
alist[[4]]
```

```
{a, b}
```

Here's the second part of that sublist:

```
alist[[4, 2]]
```

```
b
```

This gives the first and third parts:

```
alist[[{1, 3}]]
```

```
{2, y}
```

This gives the first *through* the third parts:

```
alist[[1 ;; 3]]
```

```
{2, x, y}
```

This uses `First` to extract the first part:

```
First[alist]
```

```
2
```

This uses `Last` to extract the last part:

```
Last[alist]
```

```
{a, b}
```

Creating Lists

Mathematica provides three functions that are especially useful for creating lists. These are `Range`, `Table` and `NestList`.

Range

`Range` can be used with one, two, or three arguments. With one argument, it returns a list of consecutive natural numbers beginning with 1.

```
Range[10]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

`Range[`a, b`]` returns a list containing a, $a+1$, $a+2$, …, $a+n$, where $a+n \le b < a+n+1$.

```
Range[4.5, 15.1]
```

```
{4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5, 12.5, 13.5, 14.5}
```

A third argument specifies the increment. (The default is 1.)

```
Range[0, 1, 1/10]
```

$$\left\{0, \frac{1}{10}, \frac{1}{5}, \frac{3}{10}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{7}{10}, \frac{4}{5}, \frac{9}{10}, 1\right\}$$

Table

`Table` provides an easy way of constructing many kinds of lists. The following computations illustrate its use.

```
Table[k^2, {k, 10}]
```

```
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
```

```
Table[Table[i - j, {i, 4}], {j, 4}]
```

```
{{0, 1, 2, 3}, {-1, 0, 1, 2}, {-2, -1, 0, 1}, {-3, -2, -1, 0}}
```

```
% // MatrixForm
```

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 \\ -3 & -2 & -1 & 0 \end{pmatrix}$$

■ NestList

`NestList` creates lists whose elements are terms of a *recursive sequence*; that is, given a function f and a starting point a_1, it creates a list containing $a_1, a_2, a_3, \ldots, a_n$, where $a_{k+1} = f(a_k)$. For example, ten terms of the arithmetic sequence defined by

$$a_1 = 3,\ a_{k+1} = 2a_k - 1,\ \ k = 1, 2, 3, \ldots$$

can be computed by first defining

```
f[x_] := 2 x - 1
```

and then entering

```
NestList[f, 3, 9]
```

```
{3, 5, 9, 17, 33, 65, 129, 257, 513, 1025}
```

Notice that `NestList` has three arguments. The first is the name of the function, the second is the first member of the list, and the third is the number of "steps" to be computed (which is one less than the length of the resulting list).

■ Manipulating Lists

■ Flatten

There are occasions when we need to simplify lists by "merging" smaller lists that it contains. The `Flatten` command does this. For example:

```
Flatten[{{1, 2, 3}, {4, 5}}]
```

```
{1, 2, 3, 4, 5}
```

```
Flatten[{{{x, y}, {3}}, {4, q}}]
```

```
{x, y, 3, 4, q}
```

■ Append and Prepend

We will often need to add elements to the end or beginning of an existing list. These tasks can be done with `Append` and `Prepend`. Here are two examples:

```
Append[{1, 2}, 3]
```

```
{1, 2, 3}
```

```
Prepend[{1, 2}, 0]
```

```
{0, 1, 2}
```

■ Union and Join

Merging two or more lists into one can be done with `Union` or `Join`. `Union` does *not* maintain the order of elements:

```
Union[{a, 2}, {v, 8}, {h, s}]
```

```
{2, 8, a, h, s, v}
```

but Join does:

```
Join[{a, 2}, {v, 8}, {h, s}]
```

```
{a, 2, v, 8, h, s}
```

Map and Apply

Map

A very useful method for applying a non-listable function to each element of a list is provided by Map. Suppose we have a list of ordered pairs of numbers such as

```
points = Table[{2 i, 5 - i}, {i, 8}]
```

```
{{2, 4}, {4, 3}, {6, 2}, {8, 1},
 {10, 0}, {12, -1}, {14, -2}, {16, -3}}
```

and we would like to create a list containing the sums of the numbers in each ordered pair in the list. To do this, we can create the function

```
addpairs[{x_, y_}] := x + y
```

and "map" it through the list of ordered pairs:

```
Map[addpairs, points]
```

```
{6, 7, 8, 9, 10, 11, 12, 13}
```

As an exercise, explain what goes on in the following:

```
Map[Flatten, {{3, {5, 6}}, {a, {b, c}}}]
```

```
{{3, 5, 6}, {a, b, c}}
```

Apply

Suppose that we have a function of two variables, say

```
vol[r_, h_] := π r^2 h
```

and that we would like to compute its value at a pair of numbers in a list, such as

```
measurements := {3.47, 5.12}
```

Now entering

```
vol[measurements]
```

```
vol[{3.47, 5.12}]
```

does not work. A very inconvenient, but effective, workaround is

```
vol[measurements[[1]], measurements[[2]]]
```

```
193.677
```

But a far simpler and more versatile approach is provided by Apply function:

```
Apply[vol, measurements]
```

```
193.677
```

It is usually easy to avoid such a situation (by defining `vol[{r_, h_}] := π r^2 h` in this case), but `Apply` does give us very nice way to compute the sum or product of the elements of a list:

```
Apply[Plus, {2, 5, 8, 12, 13}]
40
Apply[Times, {2, 5, 8, 12, 13}]
12 480
```

◆ Exercises

1. a) Use `Table` and `Prime` to generate a list of the first 100 prime numbers.
   b) Generate the same list using only `Prime` and `Range`.
2. Generate a list of values of the function $f(x) = \frac{\sin x}{x}$ for $x = .1, .2, \ldots, 1$, first using `Table`, then without `Table`.
3. Generate a list of the first 50 odd natural numbers, using:
   a) `Table`; b) `Range`; c) `NestList`
4. Generate a list of the first 21 powers of 2 (beginning with 2^0), using:
   a) `Table`; b) `Range`; c) `NestList`
5. Use `Table` to generate a list of ordered pairs $(x, f(x))$ for $x = 0, \frac{\pi}{12}, \frac{2\pi}{12} \ldots, \pi$, where $f(x) = \sin x$. Can you think of a way to do this without `Table`?
6. Create a list named `waves` that contains $\frac{1}{k} \sin kx$ for $k = 1, 2, 3, 4, 5$. Then plot the expressions in `waves` by entering

   ```
   Plot[Evaluate[waves], {x, 0, 2 π}]
   ```

   Now enter

   ```
   colors = Map[Hue, Range[.4, 1, .15]]
   Plot[Evaluate[waves], {x, 0, 2 π}, PlotStyle → colors]
   ```

   Then enter

   ```
   grays = Map[GrayLevel, Range[.8, 0, -.2]]
   Plot[Evaluate[waves], {x, 0, 2 π}, PlotStyle → grays]
   ```

7. a) The function

   ```
   f[x_] := x^5 - 2 x^2 - 3 x + 3
   ```

   has three real zeros. Plot its graph, and create a list named `guesses` that contains a rough estimate of each of the zeros.

   b) Define the function

   ```
   getZero[guess_] := FindRoot[f[x], {x, guess}]
   ```

   and find all three zeros of f with one command by entering

   ```
   Map[getZero, guesses]
   ```

1.5 Rules

Understanding rules is essential to making efficient use of *Mathematica*. For example, note that the `Solve` command returns its results as lists of rules:

```
soln = Solve[{x^2 + x + y^2 == 2, 2 x - y == 1}, {x, y}]
```

$$\left\{\left\{y \to \frac{1}{5}\left(-2-\sqrt{29}\right), x \to \frac{1}{10}\left(3-\sqrt{29}\right)\right\},\right.$$
$$\left.\left\{y \to \frac{1}{5}\left(-2+\sqrt{29}\right), x \to \frac{1}{10}\left(3+\sqrt{29}\right)\right\}\right\}$$

To convert this answer to a list of pairs of numbers, we apply the rules to the list `{x,y}` as follows:

```
{x, y} /. soln
```

$$\left\{\left\{\frac{1}{10}\left(3-\sqrt{29}\right), \frac{1}{5}\left(-2-\sqrt{29}\right)\right\}, \left\{\frac{1}{10}\left(3+\sqrt{29}\right), \frac{1}{5}\left(-2+\sqrt{29}\right)\right\}\right\}$$

Anticipating this in advance, we might have combined these steps by entering

```
{x, y} /. Solve[{x^2 + x + y^2 == 2, 2 x - y == 1}, {x, y}]
```

$$\left\{\left\{\frac{1}{10}\left(3-\sqrt{29}\right), \frac{1}{5}\left(-2-\sqrt{29}\right)\right\}, \left\{\frac{1}{10}\left(3+\sqrt{29}\right), \frac{1}{5}\left(-2+\sqrt{29}\right)\right\}\right\}$$

The name of the *slash-dot* object "`/.`" that we use to apply rules is `ReplaceAll`. The following are some simple examples that illustrate its use.

```
Sqrt[x^2 - x + 1] /. x → 3
```

$$\sqrt{7}$$

```
x + y /. y → x
```

```
2 x
```

```
x y + y z + x z /. {x → a, y → b + c, z → 5}
```

```
5 a + 5 (b + c) + a (b + c)
```

◆ Exercises

1. Trigonometric identities provide a good context in which to learn about rules and gain a bit of insight into symbolic computation in general. For example, the sine addition formula can be applied via the rule

```
sinAdd := Sin[x_ + y_] → Sin[x] Cos[y] + Cos[x] Sin[y]
```

Notice what happens when the rule is applied to $\sin(3x + 5y)$:

```
Sin[3 x + 5 y] /. sinAdd
```

```
Cos[5 y] Sin[3 x] + Cos[3 x] Sin[5 y]
```

The same rule provides the sine difference formula as well.

```
Sin[t - ϕ] /. sinAdd
```

```
Cos[ϕ] Sin[t] - Cos[t] Sin[ϕ]
```

This rule also handles expressions with three or more summands, provided we use `//.` (`ReplaceRepeated`) instead of `/.`:

```
Sin[a + b + c] //. sinAdd
Cos[b + c] Sin[a] + Cos[a] (Cos[c] Sin[b] + Cos[b] Sin[c])
```

a) Enter the definition of `sinAdd` and construct a similar rule, `cosAdd`, for the cosine addition formula. Test both rules on several different expressions.

b) Enter the following multiple-angle expansion formula for sine:

```
sinMult := Sin[n_Integer x_] → Sin[(n - 1) x + x] /. sinAdd
```

Check that this rule works properly by entering

```
{Sin[2 x], Sin[3 x]} //. sinMult
{2 Cos[x] Sin[x], 2 Cos[x]^2 Sin[x] + Cos[2 x] Sin[x]}
```

c) Construct a similar rule, `cosMult`, for the multiple-angle expansion formula for cosine. Test it on a few expressions.

d) Notice the result of repeatedly applying all four rules (followed by `Expand`) by entering

```
Sin[2 x + y] //. {sinAdd, cosAdd, sinMult, cosMult} // Expand
```

Then enter `Simplify[%]` to verify that the expansion is correct.

e) Define the function

```
trigExpand[expr_] :=
 expr //. {sinAdd, cosAdd, sinMult, cosMult} // Expand
```

and test it by entering

```
Cos[x + 2 y] + Sin[3 x - y] // trigExpand
% // Simplify
```

f) Finally, create a table of multiple angle formulas for sine by entering

```
Table[{Sin[k x], Sin[k x] // trigExpand}, {k, 1, 5}] // TableForm
```

and create a similar table of multiple angle formulas for cosine.

2. a) Use `NSolve` to find the zeros of the polynomial $f(x) = x^5 - 4x^4 + 12x^2 - 9x + 1$. Convert the result to a list of numbers.

b) Compute the sum of the zeros of f using `Total`. Combine the entire process into a single command.

c) Repeat the process in parts (a) and (b) after changing the coefficient of x^4 to 3, and then once again after changing the coefficient of x^4 to 1. Try changing the other coefficients to see if they affect the result. What do you conjecture about the sum of the zeros of a fifth-degree polynomial?

d) Compute the *product* of the zeros of f using `Apply` and `Times`. (See section 1.4.) Experiment with the coefficients to determine which affect the result. What do you conjecture about the product of the zeros of a fifth-degree polynomial?

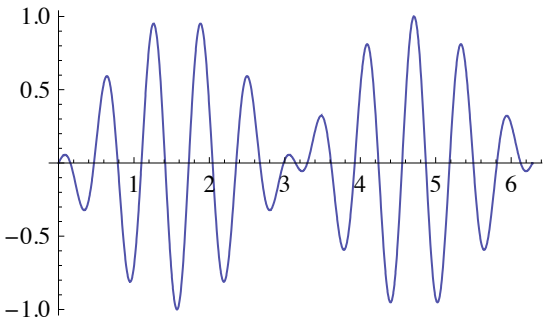
e) Experiment with a few polynomials of other degrees. Do your conjectures depend on degree? Also, are your conjectures consistent with *linear* polynomials?

1.6 Graphics

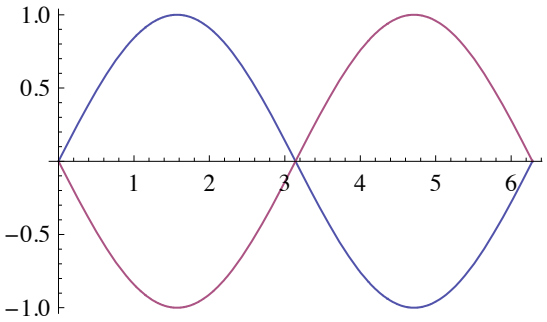
■ Graphics Objects and Show

Graphics commands such as `Plot` create and display **graphics objects**.

```
graph1 = Plot[Sin[x] Cos[10 x], {x, 0, 2 π}]
```

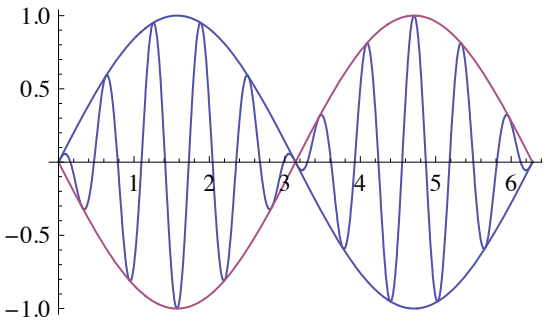



```
graph2 = Plot[{Sin[x], -Sin[x]}, {x, 0, 2 π}]
```



The `Show` command displays graphics objects, which may consist of two or more combined graphics objects.

```
Show[graph1, graph2]
```



■ Graphics Primitives

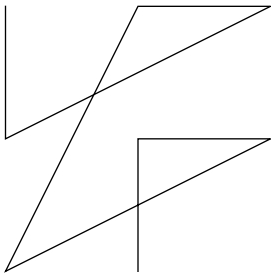
Graphics primitives are the simple objects of which more complex graphics objects are built. Two- dimensional graphics primitives include `Point`, `Line`, `Circle`, `Disk`, `Rectangle`, `Polygon`, and `Text`.

The following defines a graphics primitive consisting of a series of line sergments connecting the specified points:

```
zigzag :=
 Line[{{1, 2}, {1, 1}, {3, 2}, {2, 2}, {1, 0}, {3, 1}, {2, 1}, {2, 0}}]
```

The `Graphics` command creates a graphics object from the graphics primitive.

```
Graphics[zigzag]
```

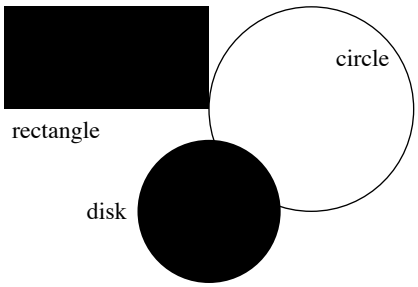


Here is a list of graphics primitives:

```
shapes = {Rectangle[{-2, 1}, {0, 2}], Circle[{1, 1}, 1],
   Disk[{0, 0}, .7], Text["rectangle", {-1.5, .8}],
   Text["disk", {-1, 0}], Text["circle", {1.5, 1.5}]};
```

This is the resulting graphics object:

```
Graphics[shapes, AspectRatio → Automatic]
```

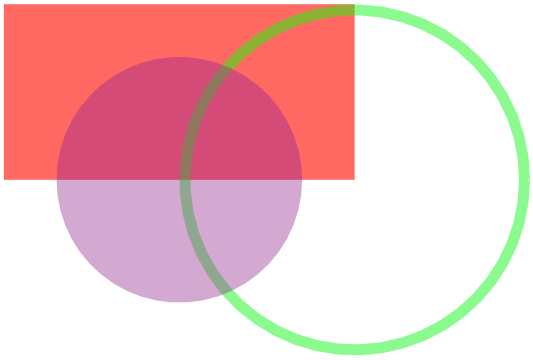


■ Graphics Directives

Graphics directives affect the way graphics primitives are displayed. Common graphics directives include `Opacity`, `PointSize`, and `Thickness`, as well as common colors such as `Red`, `Blue`, `Orange`, and so on. (`RGBColor`, `Hue`, `GrayLevel`, and `CMYKColor` are available for mixing your own colors.)

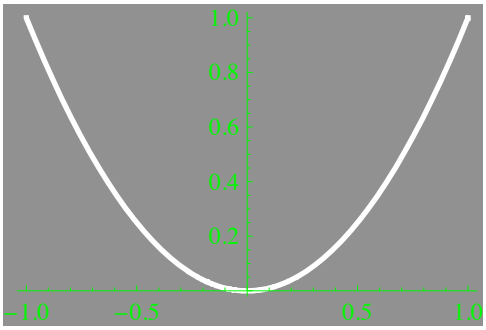
A graphics directive is associated with a graphics primitive by creating a list of the form {*directive, primitive*}. More than one primitive can be specified by creating a list of the form {*directive1, directive2, …, primitive*}. The following suggests the many possibilities:

```
redRect = {Red, Opacity[.67], Rectangle[{-1, 1}, {1, 2}]};
thickCircle =
  {Thickness[.02], Opacity[.5], Green, Circle[{1, 1}, 1]};
purpleDisk = {Purple, Opacity[.4], Disk[{0, 1}, .7]};
Graphics[{redRect, thickCircle, purpleDisk}]
```



Often graphics directives are provided through *options* such as `PlotStyle`, `AxesStyle`, and `Background`. For example,

```
Plot[x^2, {x, -1, 1}, PlotStyle → {Thick, White},
 AxesStyle → Green, Background → GrayLevel[.5]]
```



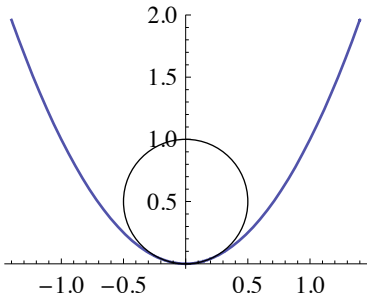
■ Suppressing and Combining Graphics

Suppose that we want to plot the parabola $y = x^2$ along with the circle of radius $1/2$ centered at $(0, 1/2)$. The following assigns names to plots of the parabola and the circle. *The output of each is surpressed by a semicolon*. (That's new in *Mathematica* 6.)

```
curve = Plot[x^2, {x, -1.4, 1.4}];
circ = Graphics[Circle[{0, .5}, .5]];
```

We can now combine the two graphics with `Show`.

```
Show[curve, circ, AspectRatio → Automatic]
```



The same thing could be accomplished all at once by entering

```
Show[Plot[x^2, {x, -1.4, 1.4}],
 Graphics[Circle[{0, .5}, .5]], AspectRatio → Automatic]
```

(This behavior is new in *Mathematica* 6; in earlier versions the preceding command would have produced three separate plots.)

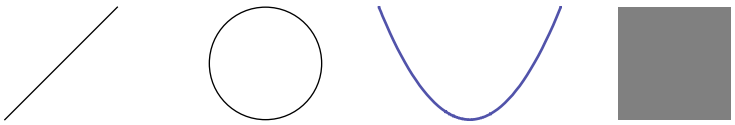
■ `GraphicsRow` and `GraphicsGrid`

Two very useful graphics commands are `GraphicsRow` and `GraphicsGrid`. With these we can create composite graphics objects containing rectangular arrays of individual graphics objects. For instance, let's create the following four graphics objects:

```
segment = Graphics[Line[{{0, 0}, {2, 2}}]];
circ = Graphics[Circle[{0, 0}, 1]];
parabola = Plot[x^2, {x, -1, 1}, Axes → None];
rect = Graphics[{Gray, Rectangle[{0, 0}, {1, 1}]}];
```

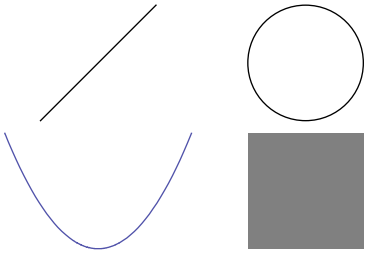
The following shows these four graphics objects in a one-by-four array:

```
GraphicsRow[{segment, circ, parabola, rect}]
```



We would get a two-by-two array instead if we enter

```
GraphicsGrid[{{segment, circ}, {parabola, rect}}]
```



◆ Exercises

1. Predict the result of each of the following commands before entering it.

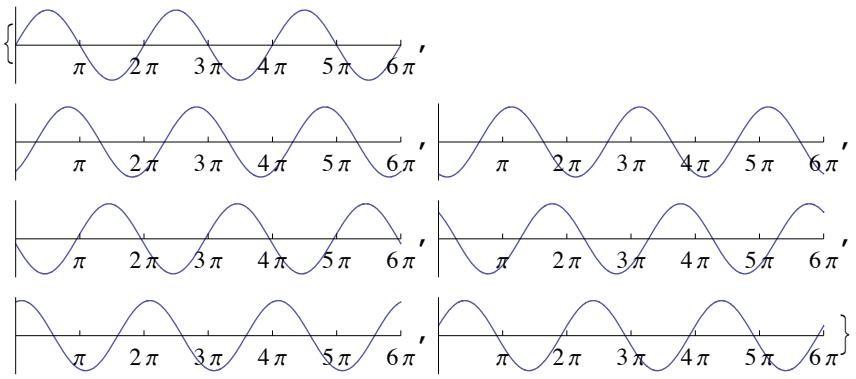
```
GraphicsRow[Table[Graphics[{Thickness[t], Orange, Circle[]}],
  {t, .02, .1, .02}]]

GraphicsRow[Table[
  Graphics[{PointSize[t], Point[{0, 0}]}], {t, .1, 1.1, .25}]]

GraphicsRow[Table[Graphics[{col, Disk[]}],
  {col, {Red, Blue, Green, Purple}}]]

GraphicsRow[Table[Graphics[{col, Disk[]}],
  {col, NestList[Lighter, Red, 3]}]]

GraphicsRow[
 Table[Graphics[{RGBColor[h, 0, 1 - h], Disk[]}], {h, 0, 1, .1}]]

Graphics[Table[{Hue[RandomReal[]],
   Disk[{Cos[t], Sin[t]}, .25]}, {t, π / 6, 2 π, π / 6}]]
```

1.7 `Animate` and `Manipulate`

One of the most instructive and fun features of *Mathematica* has always been its ability to animate graphics. New in *Mathematica* 6 is the function `Animate`, which provides a convenient, simple mechanism for creating animations.

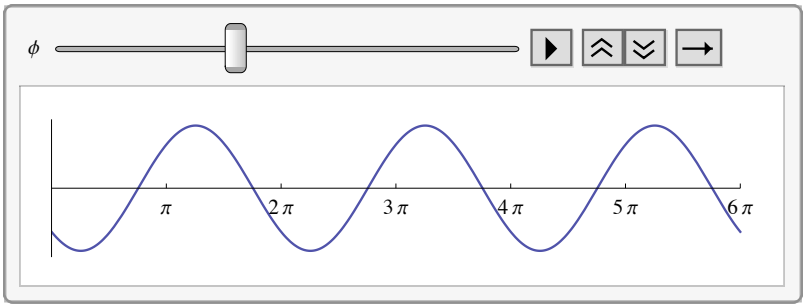
For the sake of comparison, let's first create a simple table containing graphs of $y = \sin(x - \phi)$ for various values of the phase shift ϕ. Here ϕ will go from 0 to 2π in steps of 1 (by default).

```
Table[Plot[Sin[x - ϕ], {x, 0, 6 π},
  PlotRange → {{0, 6 π}, {-1.1, 1.1}}, AspectRatio → .2,
  Ticks → {Range[6] π, None}], {ϕ, 0, 2 π}]
```



If we change `Table` to `Animate`, we get an animation instead, in which ϕ goes from 0 to 2π *continuously*. The controls allow you to start, stop, slow down, speed up, and reverse the animation. Moreover, the slider lets you "manually" move forward or backward through the animation.

```
Animate[Plot[Sin[x - ϕ], {x, 0, 6 π},
  PlotRange → {{0, 6 π}, {-1.1, 1.1}}, AspectRatio → .2,
  Ticks → {Range[6] π, None}], {ϕ, 0, 2 π}]
```



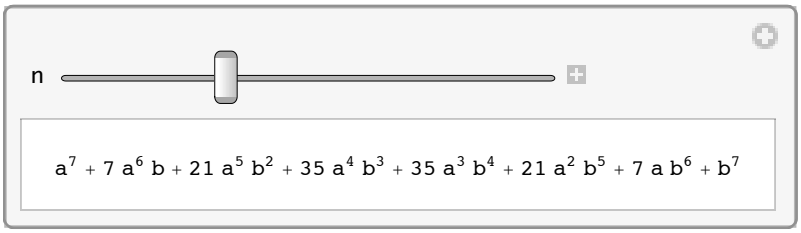
Using `Manipulate` instead, we get only the slider control. However, clicking on the ⊞ icon at the right end of the slider will reveal animation controls.

```
Manipulate[
 Plot[Sin[x - ϕ], {x, 0, 6 π}, PlotRange → {{0, 6 π}, {-1.1, 1.1}},
  AspectRatio → .2, Ticks → {Range[6] π, None}], {ϕ, 0, 2 π}]
```

💡 Notice in the preceding examples the specification of the `PlotRange` option. This is generally necessary to ensure that each plot created corresponds to the same rectangle in the plane, thereby producing an animation in which any fixed point remains still.

`Animate` and `Manipulate` can be used to animate or manipulate any type of expression, not just graphics.

```
Manipulate[Expand[(a + b)^n], {n, 1, 20, 1}] // boldEdge
```



Here are two simple examples that use a different `ControlType`:

```
Manipulate[Expand[(a + b)^n],
 {n, 1, 15, 1}, ControlType → SetterBar]
```

```
Manipulate[x^y, {x, 1, 15, 1, ControlType → SetterBar},
 {y, 1, 15, 1, ControlType → SetterBar}]
```

◆ Exercises

1. The following creats a `Manipulate` object that plots the graph of $y = \sin(a\,x) + \sin(b\,x)$ for manipulable *angular frequencies* a and b of the individual terms. Enter the command and experiment with the two sliders. Then describe what you observe whenever the value of a "crosses over" the value of b or vice versa.

```
Manipulate[Plot[Sin[a t] + Sin[b t],
  {t, 0, 5}, PlotRange → {-2, 2}, PlotPoints → 40],
 {a, 50, 100}, {{b, 100}, 50, 100}]
```

   This demonstrates the same phenomenon with sound:

```
Play[Sin[(950 + 10 t) t] + Sin[1000 t], {t, 0, 5}]
```

   And this is the same thing in stereo:

```
Play[{Sin[(950 + 10 t) t], Sin[1000 t]}, {t, 0, 5}]
```

   This is the phenomenon known as *beats*, which is useful in tuning string instruments by ear.

2. In this exercise, we will build up some simple and strangely interesting pieces of *Mathematica*-generated "art." Begin by entering

```
r := RandomReal[];
Show[Graphics[Line[{{r, r}, {r, r}}]], PlotRange → {{0, 1}, {0, 1}}]
```

   This simply plots a random line segment within the square $-1 \le x \le 1, -1 \le y \le 1$. (Re-enter this a couple of times to observe the difference in the results.) Now enter the following several times. You should observe forty random segments each time.

```
Graphics[Table[Line[{{r, r}, {r, r}}], {40}],
 PlotRange → {{0, 1}, {0, 1}}]
```

   Let's now give random color and thickness to the segments. Enter this a few times:

```
Graphics[Table[
  {RGBColor[r, r, r], Opacity[e^(-1.5 r)], Thickness[.01 + .02 r],
    Line[{{r, r}, {r, r}}]}, {40}], PlotRange → {{0, 1}, {0, 1}}]
```

   Now create a composite graphic by entering

```
GraphicsRow[Table[Graphics[Table[{RGBColor[r, r, r],
    Thickness[.003 + .01 r], Line[{{r, r}, {r, r}}]}, {40}]], {5}]]
```

   Enter the following to produce an animated piece on a black background:

```
Animate[Graphics[k; Table[
   {RGBColor[r, r, r], Opacity[.9 e^(-2 r)], Thickness[.01 + .02 r],
    Polygon[{{r, r}, {r, r}, {r, r}}]}, {30}],
  PlotRange → {{0, 1}, {0, 1}}, Background → Black], {k, 1, 20}]
```

   Repeat the animation above, replacing the `Line[{{r,r},{r,r}}]` primitive with

   a) `Circle[.25{1+2r,1+2r},7 r/5]`

   b) `Disk[{r,r}, .5r]` c) `Polygon[{{r,r},{r,r},{r,r}}]`

1.8 Avoiding and Getting Out of Trouble

A Top Eleven List: Causes of *Mathematica* Problems

11. Forgetting that the natural log function is `Log`, not `Ln`

This is not peculiar to *Mathematica*; many advanced texts use this convention. However, if this bothers you, you can always define `Ln[x_]:=Log[x]`.

10. Typing an equation with one equal sign (=) instead of two (==)

A single equal sign is used only for assignments; an equation requires two. Recovering from this mistake often requires that you `Clear` a variable.

9. Forgetting to type a space between multiplied expressions

For example, if you accidentally type `xSin[x]`, *Mathematica* assumes that you are refering to a function named `xSin`.

8. Using parentheses instead of brackets or braces (or vice-versa)

Parentheses, brackets, and braces have very specific and different uses. Parentheses are used only for grouping within expressions, brackets enclose function arguments, and braces enclose members of a list.

7. Forgetting to load a package before referencing something in it

To correct the "shadowing" problem that results from this, you can enter `Remove[`*object*`]` where *object* is what you tried to use prior to loading the package.

6. Entering a command that relies on a previous definition that has not been entered during the current session

Whenever you resume work from a previous session, be sure that you re-enter commands in order from the top of your *Mathematica* notebook.

5. Doing an enormous symbolic computation instead of a simple numerical computation

See ■ Symbolic versus Numerical Computation below.

4. Making multiple definitions for one variable or function name

See ■ Multiple Definitions and Using the Question Mark below.

3. Spelling errors (including capitalization)

Enough said.

2. Forgetting to use a `Blank` when defining a function

See Section 1.3.

1. Forgetting to save your work before the inevitable crash

A word to the wise…

■ What to Do When You Run into Trouble

- *Check for spelling mistakes, typos, and other syntax errors.*
- *Look for online help.*

  You can access the Documentation Center through the Help menu or by simply pressing your "help" key. If you want help on a particular command, option, etc., highlight that item before pressing "help."

- *Clear variable names.*

  Remember that entering `Clear[`*var1*, *var2*,…`]` clears variables.

- *Clear everything.*

  Here's a quick way to clear *all* previous definitions:

```
ClearAll["Global`*"]
```

- *Quit and restart the kernel.*

  Do this by choosing Quit Kernel: Local from your Kernel menu. You can then choose Start Kernel: Local from your Kernel menu or simply enter a command to start a new kernel session.

- *Quit and restart Mathematica.*

  Be sure to save your work first.

- *Quit Mathematica and restart your computer.*

  Again, be sure to save your work.

- *Quit and restart your day. (Just kidding.)*

  Seriously though, if you get frustrated, *take a break!*

💡 **Important note:** Although *Mathematica* remembers everything you enter during a particular session, it does not remember anything from a previous session or anything prior to clearing all variables or restarting the kernel. Since much of what you do in *Mathematica* depends on previously entered commands, you must be careful to reënter the commands that are needed after clearing all variables or restarting the kernel.

■ Interrupting Calculations

You will occasionally enter a command that takes *Mathematica* a very long time to evaluate. To stop a computation, select Abort Evaluation from the Evaluation menu. The keyboard shortcut for this is ⌘-. (command-period) on a Macintosh and CTRL-C on an Windows PC.

It is often necessary to press these keys repeated to interrupt a calculation, and sometimes there is no alternative but to quit the kernel.

■ Interpreting *Mathematica* Output When Things Don't Work

In many circumstances, *Mathematica* will give you a useful error message when a bad command is entered. Here are two examples:

```
Plot[x^2, {0, 1}]
```

```
Plot::pllim : Range specification {0, 1} is not of the form {x, xmin, xmax}. >>
```

```
Plot[x^2, {0, 1}]
```

```
Solve[x^2 + 5 x = 2, x]
```

```
Set::write : Tag Plus in 5 x + x^2 is Protected. >>
Solve::eqf : 2 is not a well-formed equation. >>
```

```
Solve[2, x]
```

However, it is very common for *Mathematica* simply to give a problematic command back to you with no message. *Mathematica* does this whenever the syntax is correct, but no currently defined rules affect the result. For example,

```
aFunctionNotEntered[0]
```

```
aFunctionNotEntered[0]
```

and

```
Ln[1]
```

```
Ln[1]
```

When *Mathematica* simply gives a command back to you with no error message, it means that the syntax is okay, but something in the command is unrecognizable.

■ Symbolic Versus Numerical Computation

It is easy to run into major trouble by inadvertently asking *Mathematica* to create a huge symbolic expression. This is most likely to happen as a result of doing some kind of recursive calculation. For example, suppose we want to calculate several terms in the sequence defined by

$$x_0 = 1 \text{ and } x_{k+1} = 3 \sin x_k - x_k \text{ for } k = 0, 1, 2, \ldots$$

Here is a typical *Mathematica* approach: We'll define the function

```
f[x_] := 3 Sin[x] - x
```

and use `NestList` to compute terms in the sequence. This computes the first five terms:

```
NestList[f, 1, 4]
```

```
{1, -1 + 3 Sin[1], 1 - 3 Sin[1] - 3 Sin[1 - 3 Sin[1]], -1 + 3 Sin[1] +
   3 Sin[1 - 3 Sin[1]] + 3 Sin[1 - 3 Sin[1] - 3 Sin[1 - 3 Sin[1]]],
 1 - 3 Sin[1] - 3 Sin[1 - 3 Sin[1]] - 3 Sin[1 - 3 Sin[1] - 3 Sin[1 - 3 Sin[1]]] -
   3 Sin[1 - 3 Sin[1] - 3 Sin[1 - 3 Sin[1]] -
      3 Sin[1 - 3 Sin[1] - 3 Sin[1 - 3 Sin[1]]]]}
```

This is not exactly what we had in mind, is it? If we had asked for ten terms instead of five, the result would have filled several pages. (Try it.) If we had asked for thirty terms, and *if* the computation had *eventually* succeeded, the result would have contained more than 1/2 *billion* copies of the expression `Sin`! (Do yourself a favor; *don't* try it.)

So what should we do? We simply need to coerce *Mathematica* into doing the calculation numerically instead of symbolically, which is what we wanted to begin with! One simple way to do this is to start the sequence with the real number `1.` instead of the integer `1`. (Can you think of two other ways to accomplish the same thing?)

```
f[x_] := 3 Sin[x] - x; NestList[f, 1., 30]
```

```
{1., 1.52441, 1.47236, 1.51312, 1.48189, 1.50626,
 1.4875, 1.5021, 1.49082, 1.49959, 1.49281, 1.49807, 1.494,
 1.49716, 1.49471, 1.49661, 1.49514, 1.49628, 1.4954,
 1.49608, 1.49555, 1.49596, 1.49564, 1.49589, 1.4957,
 1.49585, 1.49573, 1.49582, 1.49575, 1.49581, 1.49576}
```

A useful tip: When attempting a complicated computation, *start small!* In other words, see what happens when you do three steps before you try to do thirty.

■ Multiple Definitions and Using the Question Mark

Suppose that we enter

```
f[x] = x^2 + 3 x
```

$$3\,x + x^2$$

and we then realize that we forgot the `Blank` that we need to put beside the variable. So we then enter

```
f[x_] = x^2 + 3 x
```

$$3\,x + x^2$$

and everything seems fine. *Later…* when working on a different problem, we redefine `f` as

```
f[x_] = x - 2
```

$$-2 + x$$

This function behaves as we expect; we find that its graph is the expected straight line, etc. But then we enter

```
g[x_] = f[x]^2
```

$$\left(3\,x + x^2\right)^2$$

which does not give the function `g` that we expect. So what's going on here? *Mathematica* remembers our original, "erroneous" definition of the expression `f[x]`.

Using the Question Mark

To get information on any variable or other object, just type its name after a question mark. For example, to get information on `f` we'll enter

```
? f
```

```
Global`f
```

$$f[x] = 3\,x + x^2$$

```
f[x_List] := 3 x
```

```
f[x_] := Which[x ≤ -1, 1, -1 < x ≤ 1, -x, x > 1, -1]
```

```
f[x_, y_] := x + y
```

This shows us that multiple definitions are associated with `f`. In fact, we could cause *Mathematica* to associate numerous other definitions with `f`:

```
f[x_, y_] := x + y
f[x_List] := 3 x
```

Now let's get information on `f`:

```
? f
```

```
Global`f
```

$$f[x] = 3\,x + x^2$$

```
f[x_List] := 3 x
```

```
f[x_] := Which[x ≤ -1, 1, -1 < x ≤ 1, -x, x > 1, -1]
```

```
f[x_, y_] := x + y
```

When *Mathematica* encounters an expression involving `f`, it looks through the definitions associated with `f` until one makes sense for that expression. For example:

```
f[3, 5]
```

```
8
```

```
f[13]
```

```
11
```

```
f[{4, 7}]
```

```
{12, 21}
```

```
f[x]
```

$$3\,x + x^2$$

As you may well imagine, this behavior of *Mathematica* can potentially be the source of all kinds of trouble.

The key to resolving difficulties caused by multiple definitions is to `Clear` the culprit variable. If you get into a really complicated mess, try quitting the kernel or entering

```
ClearAll["Global`*"]
```

The question mark is also useful for getting the "usage message" for built-in objects. Here are a few examples:

```
? Plot
```

Plot[f, {x, x_{min}, x_{max}}] generates a plot of f as a function of x from x_{min} to x_{max}.
Plot[{f_1, f_2, ...}, {x, x_{min}, x_{max}}] plots several functions f_i. ≫

```
? $DisplayFunction
```

$DisplayFunction gives the default setting
for the option DisplayFunction in graphics functions. ≫

```
? NestList
```

NestList[f, *expr*, n] gives a list of the results of applying f to *expr* 0 through n times. ≫

```
? /.
```

expr /. *rules* applies a rule or list of rules in an
attempt to transform each subpart of an expression *expr*. ≫

■ Memory

Some of the most common difficulties that arise when using *Mathematica* are memory related—or rather, *lack-of*-memory related. *Mathematica* consists of two applications—the **kernel** and the **front end**—working together. Each of these has its own memory.

■ Kernel Memory

The kernel is the part of *Mathematica* that does the computation. Many of the computations done by *Mathematica* involve highly complex algorithms and require a great deal of memory. In addition, the kernel remembers (by default) every command entered and every computation done in a given session. So it is easy to understand why running out of kernel memory—or experiencing poor performance due to use of virtual memory—can happen.

There are a couple of simple things that you can do to conserve memory:

- Set `$HistoryLength` to some small value such as 10 (*i.e.*, enter `$HistoryLength=10`). This causes the kernel to forget older input and output lines. The default value of `$HistoryLength` is `Infinity`.
- Use the `Share` command occasionally:

```
Share[]
```

```
1 010 880
```

This causes stored expressions to "share" subexpressions, thus reducing the amount of memory used. The output shows the number of freed bytes.

Also, see ■ Symbolic versus Numerical Computation in the preceeding section.

■ Front End Memory and File Size

Front-end memory and notebook file size usually only become an issue when your notebook contains a lot of graphics. While the computations that create a graphic are done by the kernel, the code that actually produces the graphic is stored in the front end's memory.

By deleting graphics cells—especially cells containing three-dimensional graphics or graphics created with a high value for the `PlotPoints` option—you can greatly decrease the amount of front-end memory used.

When you save your work, it is information in the front end's memory that you are saving—in the form of a *Mathematica* "notebook." When a notebook becomes very large, you can usually remedy the situation by deleting graphics cells before saving. Graphics can always be reproduced, as long as the commands are saved.


A handy feature is the Delete All Output item in the Cell menu. This will let you quickly save the essence of your work in a very small file.

■ Exercises

1. Purposely commit each of the errors described in the eleven causes of problems outlined above—with the exception of numbers 4 and 1. In cases where no consequence is immediately evident, construct a subsequent scenario that exposes the error.

1.9 Turning a Notebook into a Report

You will likely be asked to put the work you do in *Mathematica* into a form that will be presentable enough to submit to your professor. Fortunately, the *Mathematica* front end serves as a very versatile word processor. In fact, this entire manual was written with *Mathematica*.

■ Cell Styles

You should always provide comments and narrative along with your calculations (whether you're using *Mathematica* or pencil and calculator). Any cell that contains text should be given Text Style by selecting Text from the Style submenu of the Format menu before you begin typing text into the cell. You can also give an existing cell Text Style by highlighting the cell bracket and selecting Text from the same menu.

You should also use Title, Section, Subsection cells, etc., to organize your notebook. These items are also in the Style submenu of the Format menu.

Aside from resulting in much nicer looking work, this is also important because when an Input cell contains text, all kinds of errors and garbage can result if it is accidentally evaluated.

```
For example notice what happens when I enter this

i enter example For happens notice this what when
```

```
or when I enter this.
```

```
Syntax::tsntxi : "this." is incomplete; more input is needed.
Syntax::tsntxi : "this." is incomplete; more input is needed.
Syntax::sntxi : Incomplete expression; more input is needed.
```

Moreover, if Input cells only contain valid *Mathematica* commands, it is possible to evalute all of them in order by selecting Evaluate Notebook from the Evaluation menu without making a big mess of your the notebook.

■ Page Breaks

Bad page breaks usually involve a large amount of blank space at the bottom of a page. Frequently this is caused by a graphic being just a little too large to fit on the current page. The default size of *Mathematica* graphics is larger than it usually needs to be for printing. So by resizing (*i.e.*, shrinking) graphics, you can avoid a lot of bad page breaks. A graphic can be resized by clicking on it and dragging a corner. Also, a good way to get smaller, consistently-sized graphics from `Plot`, for instance, is to use `SetOptions`:

```
SetOptions[Plot, ImageSize → 200]
```

How can you tell where page breaks will occur before you print? You can select Show Page Breaks from the Printing Settings submenu of the File menu. (In *Mathematica* 5.x, this is located in the Format menu.)

You can also *force* a page break between two cells by selecting Page Break from the Insert Menu.

■ Other Tips

In the Printing Settings submenu of the File menu, you'll see Printing Options... In the resulting dialog box, you can set margins and specify whether or not to print cell brackets.

In the Style Sheet submenu of the Format menu, you can choose from among several standard style sheets. The choice of style sheet affects the appearance of title and section cells, background color, etc. Experiment to find one that you like. But don't be surprised if you end up preferring the default.

◆ Exercises

1. Write a short but detailed report (2-3 pages) in *Mathematica* on any one of the following topics. Your report must include input, output, text, section, and title cells.

   a) The rational root theorem for polynomials.

   b) How to find the inverse of a one-to-one function.

   c) Even functions and odd functions.

   d) The unit circle and the graphs of sin x and cos x.

   e) The compound interest formula.

   f) Rational functions with slant asymptotes.

1.10 Miscellaneous Advice

■ The Cube Root Function

When you ask *Mathematica* for the cube root (or any odd root) of a negative number, it returns a complex number. This complex number is indeed the *principal value* the cube root function defined on the complex numbers.

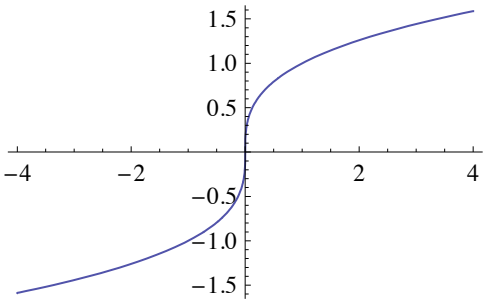
```
∛-8.
```

```
1. + 1.73205 i
```

However, this is not what we want when we talk about the cube root function defined on the real numbers. A simple remedy is to define your own cube root function as follows:

```
Cbrt[x_] := Sign[x] ∛Abs[x]
```

```
Plot[Cbrt[x], {x, -4, 4}]
```



■ Custom Initialization

You may find that there are certain commands you want to enter every time you run *Mathematica*. For instance, if you prefer that the curves drawn by `Plot` are always thicker than the default thickness, you can avoid specifying that with the `PlotStyle` option every time you use plot by entering

```
SetOptions[Plot, PlotStyle → Thickness[.005]]
```

But that will only be in effect during the current kernel session; that is, the next time you start *Mathematica*, you'll have to enter that command again.

Assuming that you have appropriate priviledges on the computer on which you're working, you can execute a group of commands automatically each time the kernel starts up by placing those commands in a file named init.m that's located in a particular directory. An easy way to find and open that file is to enter

```
ToFileName[{$UserBaseDirectory, "Kernel"}, "init.m"]
NotebookOpen[%]
```

You can type in the commands you want executed automatically upon startup and then save and close the file. Actually, it's best to type the commands in an open notebook, make sure they work correctly, and then copy and paste them into the init.m file.

Here are a few examples of things you might want to put in init.m:

• This sets options for Plot so that it draws thicker curves and makes plots 3 inches wide.

```
SetOptions[Plot,
  {PlotStyle → Thickness[Medium], ImageSize → 3 * 72}];
```

• This defines a real cube root function.

```
Cbrt[x_] := Sign[x] ∛Abs[x] ;
```

• This sets $HistoryLength to conserve memory.

```
$HistoryLength = 10
```