CHAPTER 1

Getting Started with *Mathematica*

Mathematica is a powerful software tool that integrates numerical, symbolic, mathematical, graphical, and animation capabilities. It can be used interactively to solve equations, evaluate integrals, differentiate expressions, manipulate formulae, plot graphs, and analyze data. In addition, *Mathematica* is equipped with a sophisticated programming language that supports many different programming styles. By using the built-in functions and the programming language in concert, you can effectively extend *Mathematica*'s capabilities to solve a variety of mathematical problems. Anyone who uses mathematics is sure to benefit from *Mathematica.*

Mathematica's power is derived from four independent components: the User, the Kernel, the Front End, and the Packages. Let's take a look at the responsibilities of each component.

The User

First there is you, the user, who specifies the tasks to be done or the problems to be solved. To get the most out of *Mathematica,* you need to know what commands are available, how they are used, how to write programs in *Mathematica* to perform novel tasks, and how to use the on-line help.

Fortunately, it is easy to start using *Mathematica* on simple tasks and gradually build up your skills to more complex tasks.

The Kernel

The Kernel is the computational engine that actually performs the calculations, executes commands, and computes the results. When you start up the *Mathematica* kernel, there are various functions and commands that are immediately available. These are known as the built-in functions or built-in commands. These functions are implemented as efficient C procedures, but this detail is completely hidden from the user.

The Front End

The Front End is the user interface to *Mathematica*. It is like a laboratory notebook that mediates the interaction between the User and the Kernel. The Front End also serves as a scratch pad for documenting the commands you have issued and the results you obtained. You can store commands, expressions, equations, text, graphics, sounds, and animations in a Notebook. However, unlike an ordinary paper notebook, the Front End is a hierarchically structured, active document. The hierarchical structuring refers to the fact that each notebook consists of cells that contain other cells. There are many different types of cells, each associated with their own set of properties. A Notebook is also "active" in the sense that you can issue commands to the Kernel that cause *Mathematica* to perform some calculation and place the answer back in your notebook.

Packages

Finally, there are the Packages. A Package contains a suite of functions, written in the *Mathematica* programming language, that go together naturally. *Mathematica* comes equipped with several pre-defined Packages. You should become familiar with the predefined Packages because you might find that a function you are about to write already exists in some Package.

1.1 Starting *Mathematica*

Nowadays, most users run *Mathematica* on computers that have a graphical user interface. To start the graphical user interface, find the directory containing *Mathematica* and then double click on the *Mathematica* icon (a stellate polyhedron) as shown in Figure 1.1.

Fig.1.1 Screen shot showing the *Mathematica* icon.

A window with a new Notebook will appear like that in Figure 1.2.

Fig.1.2 How a new Notebook appears on the screen in Version 3.

On computers that do not have a graphical user interface, you start *Mathematica* by typing `math` at a prompt. Typically this invokes a version of *Mathematica* that uses a command line interface. On UNIX platforms running X-windows, you can start a Notebook interface to *Mathematica* by typing `mathematica` (rather than `math`) at the prompt.

1.1.1 Remedies to Common Start-Up Problems

Mathematica is a reliable piece of software and is remarkably simple to install and run. Nevertheless, some people occasionally encounter difficulties when they try to run *Mathematica.* If you encounter problems, there is probably a simple remedy. Here are some suggestions for things to check:

1 Check that you have sufficient RAM to run *Mathematica.* In Version 3, you need at least 16 MB RAM, and realistically you should have 32 MB. In Version 2.2, you need at least 8 MB RAM, and realistically you should have 12 MB or more. If your machine has virtual memory, try setting it to what is recommended for your version of *Mathematica.* Alternatively, install more RAM on your computer.

2 If you cannot start *Mathematica* on a UNIX machine, even though you know the software ought to be available, check that the directory containing *Mathematica* is in $PATH.

3 If you cannot start *Mathematica* when you are running it over a network, check whether your software license limits the number of *Mathematica* processes that can be active at any one time. If so, ask your colleagues to exit *Mathematica* when they are not using it.

4 If you need help, you can telephone technical support at Wolfram Research, (217) 398-6500 (+44-(0)-1993-883400 in Europe) or send email to support@wolfram.com (or support-euro@wolfram.com in Europe). Be sure to have the number of your *Mathematica* license handy when you call, and include it in any email message requesting technical support. Answers to frequently asked questions (FAQs) can be found on the Wolfram Research web site, http://www.wolfram.com.

As most people run *Mathematica* via a Notebook Front End, the examples in this book assume that you are using such an interface.

1.2 The In[]'s and Out[]'s of *Mathematica*

Using *Mathematica* is like having a conversation. Ask a question and *Mathematica* responds with a result. You pose your question by typing a *Mathematica* expression in a Notebook and then sending this expression to the Kernel for evaluation. To type an input in a Notebook, you must point to the desired insertion point and click once with your mouse. If you are typing in a new Notebook, clicking the mouse creates a horizontal line across the screen that indicates where inserted text will appear. If you are using an old Notebook that already contains cells, you must click between two cells to insert a new input cell.

To have your input evaluated you need to send the expression to the Kernel. In the Notebook Front End this is accomplished by holding down the ~ key while depressing the ^ key or, alternatively, just typing the % (ENTER) key. The makers of *Mathematica* deliberately chose not to use a single carriage return to send a command to the Kernel so that users would be able to type multi-line command sequences and multi-line programs without sending each line off for evaluation.

On some computers, you might find that the first command you send to the Kernel takes an inordinate amount of time to be evaluated. Even something as simple as 2+2 can take a minute on some computers. This is because *Mathematica* loads the Kernel into memory before evaluating the command. Once loaded, however, subsequent commands are evaluated much faster. So don't be put off if *Mathematica* is slow at returning the result to your first command.

Figure 1.3 shows a screen shot after someone has plotted a function and evaluated an integral. Notice that the user has positioned the next insert point, indicated by the horizontal line, to be immediately after the integral.

Fig.1.3 Click in between cells to enter new commands.

Mathematica assigns a number in sequence for each input/output pair or exchange. The *n*-th user input in a *Mathematica* session is labeled `In[n]` and the corresponding computer output is labeled `Out[n]`. You can refer to earlier inputs and outputs by these labels. In the first exchange, which is labeled with `In[1]`, *Mathematica* is asked to evaluate 5^{10}.

In[1]:=

$$5^{10}$$

Out[1]=

```
9765625
```

As you evaluate more inputs, new input and output tags are generated.

In[2]:=

```
5!
```

Out[2]=

```
120
```

At any later time, within the same *Mathematica* session (i.e., before you terminate the Kernel process), you can refer to the *n*-th output as `%n` (or, equivalently, `Out[n]`). So `%1` (or `Out[1]`) refers to the first output, and `%2` (or `Out[2]`) refers to the second output.

In[3]:=

```
%1
```

Out[3]=

```
9765625
```

In[4]:=

```
%2
```

Out[4]=

```
120
```

When you are working with *Mathematica* interactively, you may want to refer to your most recent output. The single percent character, `%`, is used to refer to the most recent output. In the following example, `%` is replaced by the last result computed, i.e., 120.

In[5]:=

```
% - 20
```

Out[5]=

```
100
```

You can refer to previous results using multiple percent signs. That is, `%` refers to the last result, `%%` refers to the second to last result, `%%%` refers to the third to last result, etc. Any more than three percents can become confusing because the results may have scrolled off the screen. In such cases you are usually better off assigning the outputs to a variable that has a meaningful name or by referring to outputs by their explicit output number `%`*n* (for the *n*-th output, `Out[`*n*`]`). Be aware that if you restart *Mathematica,* you cannot refer to results with the numbers assigned during an earlier session. So if you have outputs that you want to save, be sure to save your Notebook before quitting *Mathematica.*

If you assign a name to a result, you can later reference that result with the assigned name. The command `mortgageRate=8` associates the value `8` with the symbol `mortgageRate`.

In[6]:=
```
mortgageRate = 8
```
Out[6]=
```
8
```

Once the name `mortgageRate` has been defined to be the number 8, you can refer to the value 8 by name.

In[7]:=
```
mortgageRate
```
Out[7]=
```
8
```

You can also perform operations by using the name of the variable. For example, to multiply the mortgage rate by 1.05 enter

In[8]:=
```
1.05 mortgageRate
```
Out[8]=
```
8.4
```

You can use names that are not assigned to numbers as a way of giving a number certain units. For example,

In[9]:=
```
2.2 lbs + 5 lbs
```
Out[9]=
```
7.2 lbs
```

So far we have discussed how you can refer to a particular output by percent notation or by assigning it to a variable. But what if you want to remind yourself what a particular input was? What do you do then? Well, just as `Out[`*n*`]` or `%`*n* refers to the *n*-th output, `In[`*n*`]` refers to the *n*-th input. Unfortunately, *Mathematica* tries to evaluate inputs immediately, so as soon as you find out what `In[`*n*`]` was, *Mathematica* evaluates it and returns `Out[`*n*`]`. For example, the first input we had was 5^{10}, but as soon as we try to grab 5^{10}, *Mathematica* evaluates it to 5^{10} = 9765625.

In[10]:=
```
In[1]
```
Out[10]=
```
9765625
```

If you want to see a particular input line without it being evaluated, convert it to a string using `InString`.

```
In[11]:=
    InString[1]
Out[11]=
    \(5^10\)
```

Although the double quotes do not appear on the screen, this output is, in fact, a string, "`\(5^10\)`", as you would see if you asked for the `FullForm[InString[1]]`. The `\(` and `\)` symbols indicate how terms are grouped in the expression.

1.3 Using On-Line Help

There are over 1500 functions, commands, options, and symbols built into *Mathematica* (as can be seen by entering the command `Length[Names["*"]]`). `Names` returns a list of all names and `Length` returns the number of elements in that list. The name of a *Mathematica* function usually indicates what it computes. Some examples are shown in Table 1.1.

Function	Description
`Binomial`	gives the binomial coefficient
`Eigenvalues`	gives a list of eigenvalues of a matrix
`FindRoot`	searches for a numerical solution to an equation
`Integrate`	evaluates the integral
`Timing`	returns the time taken to evaluate an expression

Table 1.1 Most function names are self-explanatory.

1.3.1 Using ? to Check Usage

If the name isn't enough of a clue of what a command does, you can call upon *Mathematica's* built-in on-line help to query the usage of a particular function. The ? operator is used to access information about a particular function. The symbol * used with the ? operator acts as a wild card character (i.e., it can

match any alphanumeric character or sequence of characters). If more than one command matches the request, *Mathematica* lists the names of all the commands.

Given `?Factor*` as input, *Mathematica* lists the commands that begin with the word `Factor`.

In[12]:=

```
?Factor*
```

```
Factor           FactorInteger      FactorSquareFreeList
FactorComplete   FactorList         FactorTerms
Factorial        FactorSquareFree   FactorTermsList
Factorial2
```

If only one command matches the request, *Mathematica* prints the usage statement associated with the command. The usage statement typically consists of a description of what the command does as well as a template showing how to call the command (i.e., what the command takes as an argument or as a sequence of arguments). Here is what you obtain when you ask for information on the command `FactorInteger`.

In[13]:=

```
?FactorInteger
```

```
FactorInteger[n] gives a list of the prime factors of the
integer n, together with their exponents.
```

The usage statement indicates that this command expects an integer-valued argument. Try entering `FactorInteger[75]`.

In[14]:=

```
FactorInteger[75]
```

Out[14]=

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

The result indicates that 75 is equal to .

1.3.2 Using ? and * to Find a Command

The wild card is particularly useful for browsing the built-in commands. For example, to find the names of some of the graphics commands, ask for commands whose names contain the words `Graph` or `Plot`.

In[15]:=

```
?*Graph*
```

```
ContourGraphics  FullGraphics GraphicsArray  GraphicsSpacing
```

```
DensityGraphics  Graphics     GraphicsData   SurfaceGraphics
EmbeddedGraphics Graphics3D
```

In[16]:=

```
?*Plot*
ContourPlot          MovieParametricPlot PlotDivision
DensityPlot          MoviePlot           PlotJoined
ListContourPlot      MoviePlot3D         PlotLabel
ListDensityPlot      ParametricPlot      PlotPoints
ListPlot             ParametricPlot3D    PlotRange
ListPlot3D           Plot                PlotRegion
MovieContourPlot     Plot3D              PlotStyle
MovieDensityPlot     Plot3Matrix
```

Be aware that uppercase and lowercase letters are treated as different characters.

In[17]:=

```
?*graph*
DegreeLexicographic        ParagraphIndent
DegreeReverseLexicographic ParagraphSpacing
Lexicographic              TextParagraph
```

In[18]:=

```
?*plot*
Information::nomatch :
   No symbol matching "*plot* found.
```

We looked for the keywords `*Graph*` and `*Plot*` as opposed to `*graph*` and `*plot*` because the names of *Mathematica* commands that consist of multiple words always have the first letter of each word capitalized (e.g., `MovieContourPlot`).

1.3.3 Using ?? to Obtain More Information

To find out even more information about a function, object, option, or variable, use `??`. For example, `??Plot` gives you, in addition to the usage statement, the default values of the options of `Plot`. These options specify the minimum number of points *Mathematica* samples in determining the shape of a graph, the location of the axes, the labels for the plot and the axes, and the size of the graph, among other things.

In[19]:=

```
??Plot
Plot[f, {x, xmin, xmax}] generates a plot of f as a function
of x from xmin to xmax.
Plot[{f1, f2, ... }, {x, xmin, xmax}] plots several
functions.

Attributes[Plot] = {HoldAll, Protected}

Options[Plot] =
   {AspectRatio Æ GoldenRatio^(-1), Axes Æ
    Automatic, AxesLabel Æ None, AxesOrigin Æ
    Automatic, AxesStyle Æ Automatic, Background Æ
    Automatic, ColorOutput Æ Automatic, Compiled Æ
    True, DefaultColor Æ Automatic, Epilog Æ {},
    Frame Æ False, FrameLabel Æ None, FrameStyle Æ
    Automatic, FrameTicks Æ Automatic, GridLines Æ
    None, ImageSize Æ Automatic, MaxBend Æ 10.,
    PlotDivision Æ 30., PlotLabel Æ None,
    PlotPoints Æ 25, PlotRange Æ Automatic,
    PlotRegion Æ Automatic, PlotStyle Æ Automatic,
    Prolog Æ {}, RotateLabel Æ True, Ticks Æ
    Automatic, DefaultFont ¶ $DefaultFont,
    DisplayFunction ¶ $DisplayFunction,
    FormatType ¶ $FormatType, TextStyle ¶
    $TextStyle}
```

To distinguish symbols you define from those built into *Mathematica*, you are advised to assign names that start with a lowercase letter or a $ sign. If you follow this convention, then the command ?@ lists all the objects you defined since you started up *Mathematica.*

1.3.4 Using the Help Browser

The Help Browser is a hypertext help system with links to the entire *Mathematica* reference manual, the Standard Add-On Packages manual and simple executable demonstrations of many non-trivial commands. To invoke the Help Browser, click on the “Help” menu and select “Help...” (or “Open Function Browser” if you are using an older version of *Mathematica*). A screen will pop up that lists *Mathematica* commands, Packages, and various other reference materials arranged by category. When you click on one category,

further menus of possible selections appear that allow you to focus your search down to a particular command or topic.

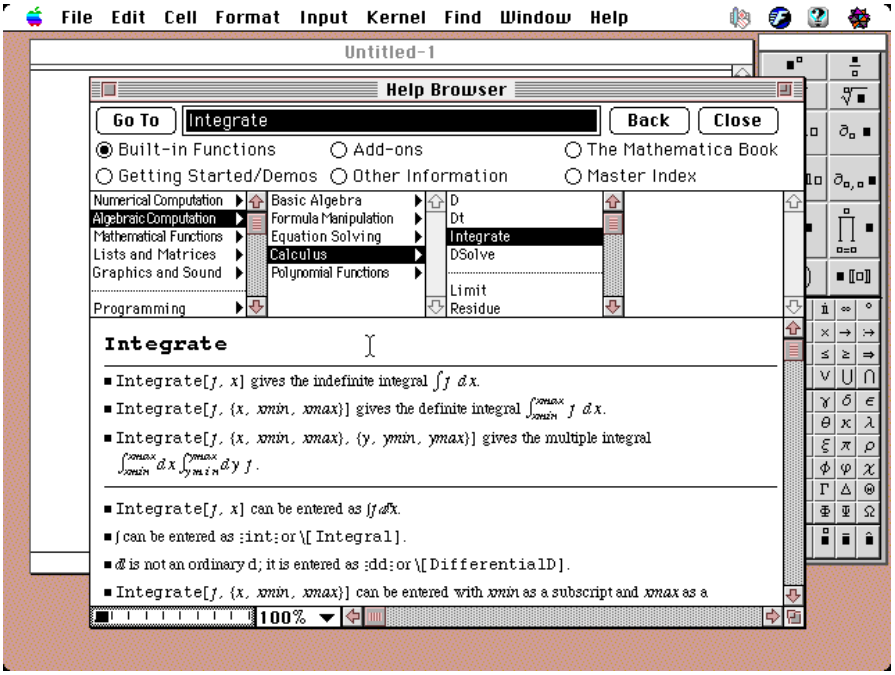

Fig.1.4 The Help Browser window in Version 3.

1.4 Using Palettes to Enter Formulae

So far we have described entering input into *Mathematica* via a one-dimensional input form. However, Figures 1.2, 1.3, and 1.4 show that the Notebook interface contains a palette of buttons on the right-hand side of the screen. This palette, and similar palettes available under the File menu, allow you to enter two-dimensional typeset mathematics.

Each palette appears as a separate window on your computer screen and holds buttons corresponding to commonly used operators such as +, commonly used characters such as Greek letters, and commonly used expression templates such as fraction templates and integral templates. The

default palette that is opened when you start *Mathematica* is shown in Figure 1.5.

Fig.1.5 The standard input palette in *Mathematica* Version 3.

As you can see, the palette contains, amongst other things, buttons for a square root template, a quotient template, an exponentiation template, a derivative template, an integration template, Greek letters (α, β, ...) and special constants (e.g., (E, , (I , and π (Pi). Many buttons have equivalent long names, e.g., $\sqrt{x} \equiv \texttt{Sqrt}[x]$, so you can type `Sqrt[x]` directly into your Notebook instead of using the square root template. To insert any of these templates in your Notebook, click in between two cells to create a new Input cell and then click on the desired button in the palette.

Templates for composite expressions contain one or more "boxes" that mark different parts of the expression. Click the box corresponding to the part of the expression that you want to enter first and type in your input. Then click on (or tab to) the other regions of the template and fill those in too. You can embed templates within templates and quickly build up an elaborate mathematical expression.

For example, the following expression can be entered using buttons on the standard input palette. Don't forget to press the ˜ and ˆ keys simultaneously, to send the expression to the kernel.

In[20]:=

Out[20]=

You can enter the same expression in a one-dimensional format.

In[21]:=

```
((1+x)-Sqrt[1-3x])/((1-x^2) (1+Sin[x]))
```

Out[21]=

The two outputs are the same regardless of how you enter the expression. So choose the style that suits you.

1.4.1 Controlling the Appearance of Outputs

You can control the appearance of an expression, *expr*, with the commands `StandardForm[`*expr*`]`, `TraditionalForm[`*expr*`]`, and `OutputForm[`*expr*`]`. Here we define *expr*.

In[22]:=

```
expr = Gamma[1 + x] Sin[x^2] / Sqrt[x^3] ;
```

The semi-colon at the end of the input line suppresses the expression being echoed back to the screen. To see *expr* formatted in `StandardForm`, we enter

In[23]:=

```
StandardForm[expr]
```

Out[23]//StandardForm=

$$\frac{\text{Gamma}[1+x]\,\text{Sin}[x^2]}{\sqrt{x^3}}$$

This is aesthetically pleasing but still uses *Mathematica*'s representation of the gamma and sine functions. To see *expr* formatted in `TraditionalForm`, we enter

In[24]:=

```
TraditionalForm[expr]
```

Out[24]//TraditionalForm=

$$\frac{\Gamma(x+1)\,\sin(x^2)}{\sqrt{x^3}}$$

`TraditionalForm` is as close as possible to conventional mathematical notation. Finally, `OutputForm` is the formatting used in earlier versions of *Mathematica*.

In[25]:=

```
OutputForm[expr]
```

Out[25]//OutputForm=

```
Gamma[1+x]Sin[x^2]
------------------
     Sqrt[x^3]
```

Notice that certain symbols, such as square root signs, are not formatted correctly and superscripts appear to be larger than is conventional. There is no reason to use `OutputForm` since `StandardForm` and `TraditionalForm` are usually more pleasing to the eye.

You can make all the expressions that will be outputs in your Notebook be formatted in `StandardForm` by clicking on the Cell≈Default Output FormatType ≈StandardForm menu (where the symbols to the right of the ≈ signs are sub-menus of those to the left of the ≈ signs). The Cell menu appears along the top edge of all Notebook windows. The Default Output FormatType sub-menu is under the Cell menu. Once selected, the default remains in effect until it is reset or until you issue a command that explicitly requests that a particular output be formatted in some other form.

1.5 Notation Conventions

Once you grasp a few key details of *Mathematica*'s notation conventions, you will be able to make more sense of *Mathematica* outputs and usage statements, and you will be able to guess the name of an unfamiliar *Mathematica* command to perform some desired function.

Typically, *Mathematica* uses complete English words for function names. However, there are a few exceptions to this rule.

1.5.1 Standard Symbols

Most people are used to referring to standard mathematical functions with symbols, rather than words (i.e., symbols such as +, –, *, /, <, and > rather than the words `Plus`, `Minus`, `Times`, `Divide`, `Less`, and `Greater`). Some examples are shown in Table 1.2.

Symbol	Full Name
+	`Plus`
-	`Minus`
*	`Times`
/	`Divide`
^	`Power`
! (in n!)	`Factorial`
<	`Less`
<= or £	`LessEqual`
>	`Greater`
>= or ≥	`GreaterEqual`

Table 1.2 Special prefix, infix, and postfix operators.

Mathematica translates an expression into function calls; so you can use many of the mathematical symbols to which you have grown accustomed. Here are examples that use some of the built-in mathematical symbols:

In[26]:=

```
4+7
```

Out[26]=

```
    11
```

In[27]:=

```
    4/2
```

Out[27]=

```
    2
```

In[28]:=

```
    3 < 6
```

Out[28]=

```
    True
```

Just as you can ask *Mathematica* about functions, you can ask about mathematical symbols or other special forms.

In[29]:=

```
    ?<
    x < y yields True if x is determined to be less than y. x1 <
    x2 < x3 yields True if the xi form a strictly increasing
    sequence.
```

Mathematical symbols and other special forms are aliases for functions built into *Mathematica*. If you ever forget what a special form represents, you can use the function `Alias` to tell you the function that corresponds to a special form.

In[30]:=

```
    Alias["<"]
```

Out[30]=

```
    Less
```

In[31]:=

```
    Alias["="]
```

Out[31]=

```
    Set
```

In[32]:=

```
    Alias["/."]
```

Out[32]=

```
    ReplaceAll
```

The inside front cover of this book lists many of the mathematical symbols that can be used in your *Mathematica* input.

1.5.2 Parentheses, Brackets, and Braces

In *Mathematica,* parentheses, square brackets, and curly braces are used for different purposes. It is important to learn these differences quickly.

Parentheses for Grouping

Parentheses are used for grouping expressions. Without parentheses, multiplication and division have a higher precedence than addition and subtraction. Thus `1+2*3` is interpreted as `1+(2*3)`.

In[33]:=
```
1 + 2 * 3
```
Out[33]=
```
7
```

Similarly, `1/2-5` is interpreted as `(1/2)-5`.

In[34]:=
```
1 / 2 - 5
```
Out[34]=

By using parentheses, you can change the grouping of arguments.

In[35]:=
```
(1 + 2) * 3
```
Out[35]=
```
9
```

In[36]:=
```
1 / (2 - 5)
```
Out[36]=

Parentheses can help to make your code more comprehensible.

Square Brackets for Function Arguments

Square brackets are used for specifying arguments to functions. For example, the function `Divisors` takes a single argument, an integer, as you can see from the built-in on-line help message.

In[37]:=
```
?Divisors
Divisors[n] gives a list of the integers that divide n.
```

The opening square bracket [separates the head of the function (the symbol made up from the characters D-i-v-i-s-o-r-s) from the argument (*n*, an integer). The closing right square bracket,], shows where the sequence of arguments ends. When you call Divisors with an integer argument, *Mathematica* returns a list of all the integers that divide the number exactly.

In[38]:=
```
Divisors[100]
```
Out[38]=
```
{1,2,4,5,10,20,25,50,100}
```

Some functions, such as Random[], do not require any argument. This is because Random makes an assumption if no argument is provided. Random called with no arguments is interpreted as a request for a pseudo-random real number in the range 0 to 1.

In[39]:=
```
?Random
Random[ ] gives a uniformly distributed pseudorandom Real in
the range 0 to 1. Random[type, range] gives a pseudorandom
number of the specified type, lying in the specified range.
Possible types are: Integer, Real and Complex. The default
range is 0 to 1. You can give the range {min, max}
explicitly; a range specification of max is equivalent to {0,
max}.
```

In[40]:=
```
Random[]
```
Out[40]=
```
0.426495
```

As you can see from the usage statement, you can also call Random with one argument (a type of number) or two arguments (a type of number and a range; i.e., a list consisting of two numbers).

In[41]:=
```
Random[Integer]
```
Out[41]=
```
1
```

In[42]:=
```
Random[Integer, {50, 60}]
```
Out[42]=
```
53
```

Be aware that some functions, like the `Sin` function, take a fixed number of arguments, while others, like `Random`, can be called with either no arguments (`Random[]`) or a variable numbers of arguments (`Random[Integer,{`*m*, *n*`}]`, `Random[`*distribution*`]`).

Braces for Lists, Nested Lists, Vectors, and Matrices

Curly braces are used for specifying lists, vectors, and matrices (arrays). A list or a vector is a sequence of one or more expressions separated by commas and enclosed in braces.

In[43]:=
```
{„, Tan[x], }
```
Out[43]=

The symbol „ is the new *Mathematica* symbol for the base of a natural logarithm. This symbol is identical to `E`. Notice that a list can contain different types of expressions.

In *Mathematica*, a matrix is represented as a list of lists. You can construct a matrix of any dimension. The following is a 2 (2 matrix whose elements are `a[`*i*, *j*`]`, where *i* specifies the row of the matrix element and *j* the column. This matrix can be entered by typing each row of the matrix at the keyboard.

In[44]:=
```
{{a[1,1], a[1,2]},
 {a[2,1], a[2,2]}}
```
Out[44]=
```
{{a[1,1],a[1,2]},{a[2,1],a[2,2]}}
```

Notice how the matrix is printed on the screen. To see a matrix in a more conventional format, use `MatrixForm`. Using a single percent sign to refer to the last output, we can apply `MatrixForm` to our matrix:

In[45]:=
```
MatrixForm[%]
```
Out[45]//MatrixForm=

Double Square Brackets for Part Specification

Double square brackets are used for denoting a part of an expression. Suppose you have the list (or vector) *v* as shown here:

In[46]:=
```
v = {a, b, c}
```
Out[46]=
```
{a,b,c}
```

The notation *v*`[[`*i*`]]` (two left square brackets followed by an integer and two right square brackets) or *v*P*i*T (entered using a template button on the BasicInput palette) returns the *i*-th element in the list (or vector) called *v* if *i* is an integer between 1 and the number of elements in the list. So the second element is given by

In[47]:=
```
vP2T
```
Out[47]=
```
b
```

You can also obtain the zero-th element of the list, which is known as the head of the expression, by calling *v*P0T. In fact, you can obtain the head of any *Mathematica* expression, *expr*, by calling *expr*P0T or `Head[`*expr*`]`. The `Head` of a list, {*a, b, c*}, is the symbol made up from the letters `L`, `i`, `s`, and `t` because internally a list is stored as the data structure `List[`{*a, b, c*}`]`.

In[48]:=
```
vP0T
```
Out[48]=
```
List
```

In[49]:=
```
Head[v]
```
Out[49]=
```
List
```

Be aware that *Mathematica* starts numbering elements in a vector at 1, unlike the C programming language, which starts at 0.

As was mentioned before, in *Mathematica,* a matrix is represented as a list of lists. Let's assign the name *m* to a 2 (3 matrix.

In[50]:=
```
m = {{1,2,3},
     {4,5,6}}
```
Out[50]=
```
{{1,2,3},{4,5,6}}
```

In[51]:=

```
In[51]:=
MatrixForm[m]
Out[51]//MatrixForm=
```

You can extract the *i*-th row of the matrix *m* by using *m*P*i*T if *i* is an integer between 1 and the number of rows in *m*.

```
In[52]:=
m[[2]]
Out[52]=
{4,5,6}
```

Similarly, the notation *m*P*i*,*j*T returns the *j*-th element in the *i*-th row.

```
In[53]:=
m[[2,1]]
Out[53]=
4
```

So by using double square brackets, you can select specific elements from a list, vector, or matrix. Double square brackets (the shorthand notation for `Part`) are extremely useful.

Comments

Text enclosed between `(*` and `*)` is not evaluated. It is interpreted as a comment.

```
In[54]:=
2 + 7 (* This is a comment *)
Out[54]=
9
```

1.5.3 The Names *Mathematica* Uses for Functions

For the most part, *Mathematica* uses complete names for functions. This convention helps you to guess the name *Mathematica* uses for a particular function if you know what it is called in conventional mathematical language. The names of all functions, variables, options, and constants built into *Mathematica* start with capital letters (e.g., `Integrate`, `Plot`). If a name consists of two or more words, the first letter or each word is capitalized (e.g., `ContourPlot`, `InterpolatingPolynomial`, `MapAt`).

Most names of objects built into *Mathematica* are complete words. Abbreviations, such as those shown in Table 1.3, are used only when they are extremely well known.

Function	Description
`Abs`	Gives the absolute value of a number.
`Cos`	Gives the trigonometric function cosine.
`D`	Computes the derivative.
`Det`	Calculates the determinant of a matrix.
`GCD`	Computes the greatest common divisor.

Table 1.3 Some function names are well-known abbreviations.

1.6 Saving Your Work and Quitting

To save the contents of a Notebook, you should click on the File menu in the Notebook interface and select the Save option on the menu that will appear. The first time you save a Notebook, *Mathematica* will prompt you for a destination. After that, *Mathematica* will save your Notebook using the file name and the file location you picked. Note that the inputs, outputs and input/output numbering disappear when you terminate a Notebook session. Results are not written permanently into the Notebook unless you explicitly tell *Mathematica* to do so (using Save).

To quit a Notebook, go to the File menu and select Quit (the bottom entry).

1.7 A Quick Tour of Some Commands

There are certain *Mathematica* commands that tend to be particularly useful in practical problem solving. As a quick way to get started with *Mathematica*, try running a few of these commands now. The commands that we shall introduce are `N` (for forcing numerical evaluation), `Table` (for creating lists), `Part` (for accessing a particular element of a list), `Plot` (for drawing graphs of

functions), `ListPlot` (for plotting data), and `Integrate` (for performing symbolic integration). The following tour provides examples of how to use each of these commands. To make *Mathematica* evaluate an input, you must position the cursor over the cell containing the desired input, click the mouse, and then hold down the shift key while you hit the return key.

Here we ask for the numerical value of the square root of 13:

In[55]:=
```
N[]
```
Out[55]=
```
3.60555
```

Note that can be entered either via the square root button on the BasicInput palette or by typing the full name `Sqrt[13]`.

The next command produces a list of the first 10 Fibonacci numbers. `Table` can be called with two arguments, an expression that depends on a variable, *i*, and the range of values to use for *i*.

In[56]:=
```
fibSequence = Table[Fibonacci[i], {i, 10}]
```
Out[56]=
```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

To extract the seventh element of the previous list, use `Part`:

In[57]:=
```
Part[fibSequence, 7]
```
Out[57]=
```
13
```

Or, equivalently, you can use `fibSequence`P7T.

In[58]:=
```
fibSequenceP7T
```
Out[58]=
```
13
```

You can draw the graph of a function using the `Plot` command. For example, to draw the graph of $\sin(x) - \sin(2x)$ for x going from 0 to 6π, you would enter the following:

In[59]:=
```
Plot[Sin[x] - Sin[2 x], {x, 0, 6π}];
```

Fig. 1.6 A plot of a function.

To plot data, use the command `ListPlot`. For example, here is a list of five pairs of numbers {{1,1},{2,4},{3,9},{4,16},{5,25}}. To plot this data so that the data points are joined together with straight line segments, we would enter

In[60]:=
```
ListPlot[{{1,1}, {2,4}, {3,9}, {4,16}, {5,25}},
        PlotJoined Æ True];
```

Fig. 1.7 A plot of some data.

To integrate an expression, use `Integrate`:

In[61]:=
```
Integrate[x Sin[x²], x]
```
Out[61]=

Note that the term x^2 can be entered using either the ‡ button on the BasicInput palette or else as x^2.

1.8 Using the Standard Packages

Most of *Mathematica* is written in the C programming language. However, certain functions, usually of a more specialized nature, are written in *Mathematica*'s own programming language and stored in Packages. A standard set of Packages came with your copy of *Mathematica.* So you have a lot of useful programs at your finger tips covering topics such as linear algebra, matrix manipulation, Laplace transforms, continuous and discrete distributions, hypothesis testing, and advanced graphics. It is well worth familiarizing yourself with what's available in the Packages. If possible, you should obtain a copy of "The Guide to Standard Packages," written by Wolfram Research. This book is available in bookstores.

Before you can use any of the functions in a Package, you must load the Package. Thus, suppose that you want to use the function `Mean` defined in the standard add-on Package "`Statistics`DescriptiveStatistics`"`. The `Mean` function computes the average of a set of numbers. To load a Package, use the `Needs` command or the `Get` (`<<`) command like so:

In[62]:=
```
Needs["Statistics`DescriptiveStatistics`"]
```

In[63]:=

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

Out[63]=

```
5.5
```

After loading a Package you can use the ? command to list the commands that it makes available. For example, to see the contents of "Statistics`DescriptiveStatistics`", you could evaluate the command `?Statistics`DescriptiveStatistics`*"`. Be sure to load the Package before entering the query.

1.9 Summary

This chapter should get you started using *Mathematica.* You have seen how to give an instruction to *Mathematica*, how to reference previous results, how to assign a result to a variable, and how to access the on-line help. The best way to learn *Mathematica* is to start using it to do simple tasks and gradually build your way up to more complicated tasks. Start with things that are familiar and whose answers you can predict. Then take some chances and experiment. Use the on-line help browser or the ? command to look up functions that your might find useful. There are several help features built into *Mathematica* that you can turn to if you get stuck. These are summarized in Table 1.4.

Command	Description
?*functionName*	Show usage for a function.
??*functionName*	Show extra information about a function.
?*xyz**	List objects whose names begin with *xyz*.
?**xyz**	List objects whose names contain *xyz*.
?@	List objects whose names consist of non-uppercase letters.
?"*context`packageName`**" Click on Help on toolbar	List all objects in a Package. Start the Help Browser.

Table 1.4 Obtaining information interactively.

Pay careful attention to the shapes of the brackets used in expressions. Table 1.5 provides a summary of the key distinctions.

Bracket	Purpose	Examples
(*expr*)	Grouping	(a + b)/(c + d)
f[*expr*]	Arguments of a function	Sin[ω/(2 π)]
{*a*, *b*, *c*}	List	{x, 2 x, 3 x}
{{*a*,*b*}, {*c*,*d*}}	Matrix	{{1, ,}, {0, 1}}
{{*a*}, {*b*}, {*c*}}	Column vector	{{0}, {1}, {0}}
{{*a*,*b*,*c*}}	Row vector	{{0,1,0}}
*list*P *i* T	*i*-th element of a list	v[[2]] or vP2T
*matrix*P*i* , *j*T	*ij*-th element of a matrix	m[[1,2]] or mP1,2T
(* *comment* *)	Commenting	(* watch out *)

Table 1.5 Shapes of brackets and their possible meanings.

Finally, remember that a great deal of *Mathematica*'s power lies in the functions defined in the add-on Packages. To make use of any of the functions in a Package you must first load the Package using the Needs or Get (i.e., <<) commands:
Needs[*"context`packageName`"*] or
<<*context`packageName`*
We will explain more about Packages and how to make them load automatically as needed later.

1.10 Exercises

1.1 Use the ? command to

(a) Find all commands whose names begin with the letter O.

(b) Find all commands that have the word List in their names.

1.2 Evaluate the following expressions using *Mathematica.* Make sure you understand the difference between (a) and (b) and between (c) and (d):

(a) 3 *x* + *x* 7

(b) 3*x* + *x*7

(c) 2 meters + 13 meters

(d) 1 mile + 7 miles

(e) 120 lbs (1 kg / 2.2 lbs)

1.3 Load the package "`Graphics`Graphics`".

(a) Use the following command to make a pie chart showing the percentages of *Mathematica* users by job title.

```
PieChart[{ {31, "Research"},
           {26, "Prof"},
           {15, "Eng"},
           { 8, "Student"},
           {13, "CS"},
           { 7, "Other"}  }];
```

(b) Use the on-line help to look up the usage of `PieChart`. Re-plot the pie chart with the wedge for the "Student" category cut away from the center of the chart to give it added emphasis.

1.4 In Version 3, use the Help menu to display the following information:

(a) The usage page for the built-in function `ListPlot`.

(b) The "Tour of *Mathematica*" pages.

(c) The description of the enhanced `Limit` function defined in the standard add-on package "`Calculus`Limit`".

1.5 Use the BasicInput palette, available under the File menu as File≈Palettes≈BasicInput, where the symbol ≈ indicates a sub-menu, to enter two-dimensional *Mathematica* versions of the following formulae:

(a) $x^{\sin(x)}$

(b) $\int_0^\infty \exp(-\alpha x^2)\,dx$

(c) $\sum_{i=0}^{n} \frac{1}{x^i}$

(d) $\frac{d}{dx}\left(\sqrt{x}\sin^{-1}(x)\right)$

(e) $\cos\left(45^\circ\right)$

(f) $\lim_{x \to 0} \frac{\sin(x)}{x}$

1.6 Use the `Needs` command to load the standard add-on Package "`Statistics`DescriptiveStatistics`". List all the functions defined in this package. Use one of them, the `Median` function, to calculate the median value of the set of numbers {1, 1, 2, 3, 5, 8, 13, 21, 34, 55}.

1.7 Try out the following commands in the order shown. They illustrate the use of the percent sign (%) to refer to previous outputs and the use of double square brackets (`[[ ]]` or P T) to refer to parts of expressions. The command `Plot[`*f*, {*x, xmin, xmax*}`]` creates a two-dimensional graph of the function *f* over the interval *x* = *xmin* to *x* = *xmax* (inclusive). `Show[`*plot1, plot2*`]` creates a single graphic that combines *plot1* and *plot2*. The function `Prime[`*n*`]` returns the *n*-th prime number. The function `Table` creates a list of values of some function, sampled at discrete points.

(a) `Plot[Sin[x], {x, 0, 10}]`

(b) `Plot[Cos[x]^2, {x, 0, 10}]`

(c) `Show[%, %%]`

(d) `Table[Prime[x], {x, 1, 10}]`

(e) `%[[5]]`

Comment. Notice that the `Plot` and `Table` commands attach different meanings to the construct {*x, xmin, xmax*}. `Plot` treats the construct as a continuous interval between *xmin* and *xmax*. `Table` treats the construct as an iterator that cycles from *x* = *xmin* to *x* = *xmax* in steps of 1. Note also that the output referenced by % changed between question (c) and question (e). The single percent character, %, always refers to the *last* expression evaluated.

1.8 Use the `Plot` and `ListPlot` commands to generate the figures (a) through (d) below. In each case, the function to be plotted can be obtained using only the sine, cosine, exponential, and polynomial functions (i.e., `Sin`, `Cos`, `Exp`, *x*^*i*). However, you might need to embed these functions inside `Table`, `Max`, and `If` commands.

(a)

(b)

(c)

(d)