2 Mathematics with Maple: The Basics

This chapter introduces the Maple commands necessary to get you started. Use your computer to try the examples as you read.

In This Chapter

- Exact calculations
- Numerical computations
- Basic symbolic computations and assignment statements
- Basic types of objects
- Manipulation of objects and the commands

Maple Help System

At various points in this guide you are referred to the Maple help system. The help pages provide detailed command and topic information. You may choose to access these pages during a Maple session. To use the help command, at the Maple prompt enter a question mark (`?`) followed by the name of the command or topic for which you want more information.

```
?command
```

2.1 Introduction

This section introduces the following concepts in Maple.

- Semicolon (`;`) usage

- Representing exact expressions

The most basic computations in Maple are numeric. Maple can function as a conventional calculator with integers or floating-point numbers. Enter the expression using natural syntax. A *semicolon* (;) marks the end of each calculation. Press ENTER to perform the calculation.

```
> 1 + 2;
```

$$3$$

```
> 1 + 3/2;
```

$$\frac{5}{2}$$

```
> 2*(3+1/3)/(5/3-4/5);
```

$$\frac{100}{13}$$

```
> 2.8754/2;
```

$$1.437700000$$

Exact Expressions

Maple computes exact calculations with rational numbers. Consider a simple example.

```
> 1 + 1/2;
```

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

The result of $1 + 1/2$ is $3/2$ not 1.5. To Maple, the rational number $3/2$ and the floating-point approximation 1.5 are distinct objects. The ability to *represent exact expressions* allows Maple to preserve more information about their origins and structure. Note that the advantage is greater with more complex expressions. The origin and structure of a number such as

$$0.5235987758$$

are much less clear than for an exact quantity such as

$$\frac{1}{6}\pi$$

Maple can work with rational numbers and arbitrary expressions. It can manipulate integers, floating-point numbers, variables, sets, sequences, polynomials over a ring, and many more mathematical constructs. In addition, Maple is also a complete programming language that contains procedures, tables, and other programming constructs.

2.2 Numerical Computations

This section introduces the following concepts in Maple.

- Integer computations
- Continuation character (\)
- Ditto operator (%)
- Commands for working with integers
- Exact and floating-point representations of values
- Symbolic representation
- Standard mathematical constants
- Case sensitivity
- Floating-point approximations
- Special numbers
- Mathematical functions

Integer Computations

Integer calculations are straightforward. Terminate each command with a semicolon.

```
> 1 + 2;
```

$$3$$

```
> 75 - 3;
```

$$72$$

```
> 5*3;
```

$$15$$

```
> 120/2;
```

$$60$$

Maple can also work with arbitrarily large integers. The practical limit on integers is approximately 2^{28} digits, depending mainly on the speed and resources of your computer. Maple can calculate large integers, count the number of digits in a number, and factor integers. For numbers, or other types of continuous output that span more than one line on the screen, Maple uses the *continuation character* (\) to indicate that the output is continuous. That is, the backslash and following line ending should be ignored.

```
> 100!;
```

```
9332621544394415268169923885626670049071\
1596826438162146859296389521759999322991\
5608941463976156518286253697920827223758\
2511852109168640000000000000000000000000\
00000
```

```
> length(%);
```

$$158$$

This answer indicates the number of digits in the last example. The *ditto operator*, (%), is a shorthand reference to the result of the previous computation. To recall the second- or third-most previous computation result, use %% and %%%, respectively.

Table 2.1 Commands for Working with Integers

Function	*Description*
`abs`	absolute value of an expression
`factorial`	factorial of an integer
`iquo`	quotient of an integer division
`irem`	remainder of an integer division
`iroot`	approximate integer root of an integer
`isqrt`	approximate integer square root of an integer
`max`, `min`	maximum and minimum of a set of inputs
`mod`	modular arithmetic
`surd`	real root of an integer

Commands for Working With Integers

Maple has many commands for working with integers, some of which allow for calculations of the factorization of an integer, the greatest common divisor (gcd) of two integers, integer quotients and remainders, and primality tests. See the following examples, as well as Table 2.1.

```
> ifactor(60);
```

$$(2)^2\ (3)\ (5)$$

```
> igcd(123, 45);
```

$$3$$

```
> iquo(25,3);
```

$$8$$

```
> isprime(18002676583);
```

$$true$$

Exact Arithmetic—Rationals, Irrationals, and Constants

Maple can perform exact rational arithmetic, that is, work with rational numbers (fractions) without reducing them to floating-point approximations.

```
> 1/2 + 1/3;
```

$$\frac{5}{6}$$

Maple handles the rational numbers and produces an exact result. The distinction between *exact* and *approximate* results is important. The ability to perform exact computations with computers enables you to solve a range of problems. Maple can produce floating-point estimates. Maple can work with floating-point numbers with many thousands of digits, producing accurate estimates of exact expressions.

```
> Pi;
```

$$\pi$$

```
> evalf(Pi, 100);
```

```
3.1415926535897932384626433832795028841\
9716939937510582097494459230781640628\
6208998628034825342117068
```

Maple distinguishes between *exact and floating-point representations of values*. Here is an example of a rational (exact) number.

```
> 1/3;
```

$$\frac{1}{3}$$

The following is its floating-point approximation (shown to ten digits, by default).

```
> evalf(%);
```

$$0.3333333333$$

These results are not the same mathematically, and they are not the same in Maple.

Important: Whenever you enter a number in decimal form, Maple treats it as a floating-point approximation. The presence of a decimal number in an expression causes Maple to produce an approximate floating-point result, since it cannot produce an exact solution from approximate data. Use floating-point numbers when you want to restrict Maple to working with non-exact expressions.

```
> 3/2*5;
```

$$\frac{15}{2}$$

```
> 1.5*5;
```

$$7.5$$

You can enter exact quantities by using *symbolic representation*, for example, π in contrast to 3.14. Maple interprets irrational numbers as exact quantities. Here is how you represent the square root of two in Maple.

```
> sqrt(2);
```

$$\sqrt{2}$$

Here is another square root example.

```
> sqrt(3)^2;
```

$$3$$

Maple recognizes the *standard mathematical* constants, such as π and the base of the natural logarithms, e. It works with them as exact quantities.

```
> Pi;
```

$$\pi$$

```
> sin(Pi);
```

$$0$$

The exponential function is represented by the Maple function `exp`.

```
> exp(1);
```

$$e$$

```
> ln(exp(5));
```

$$5$$

The example with π may look confusing. When Maple is producing typeset real-math notation, it attempts to represent mathematical expressions as you might write them yourself. Thus, you enter π as `Pi` and Maple displays it as π.

Maple is *case sensitive.* Ensure that you use proper capitalization when stating these constants. The names `Pi`, `pi`, and `PI` are distinct. The names `pi` and `PI` are used to display the lowercase and uppercase Greek letters π and Π, respectively. For more information on Maple constants, enter `?constants` at the Maple prompt.

Floating-Point Approximations

Maple works with exact values, but it can return a floating-point approximation up to about 2^{28} digits, depending upon your computer's resources. Ten or twenty accurate digits in floating-point numbers is adequate for many purposes, but two problems severely limit the usefulness of such a system.

- When subtracting two floating-point numbers of almost equal magnitude, the relative error of the difference may be very large. This is known as *catastrophic cancellation.* For example, if two numbers are identical in their first seventeen (of twenty) digits, their difference is a three-digit number accurate to only three digits. In this case, you would need to use almost forty digits to produce twenty accurate digits in the answer.

- The mathematical form of the result is more concise, compact, and convenient than its numerical value. For instance, an exponential function provides more information about the nature of a phenomenon than a large set of numbers with twenty accurate digits. An exact analytical description can also determine the behavior of a function when extrapolating to regions for which no data exists.

The `evalf` command converts an exact numerical expression to a floating-point number.

```
> evalf(Pi);
```

$$3.141592654$$

By default, Maple calculates the result using ten digits of accuracy, but you can specify any number of digits. Indicate the number after the numerical expression, using the following notation.

```
> evalf(Pi, 200);
```

```
3.1415926535897932384626433832795028841\
97169399375105820974944592307816406286\
20899862803482534211706798214808651328\
23066470938446095505822317253594081284\
81117450284102701938521105559644622948\
9549303820
```

You can also force Maple to do all its computations with floating-point approximations by including at least one floating-point number in each expression. Floats are *contagious*: if an expression contains one floating-point number, Maple evaluates the entire expression using floating-point arithmetic.

```
> 1/3 + 1/4 + 1/5.3;
```

$$0.7720125786$$

```
> sin(0.2);
```

$$0.1986693308$$

The optional second argument to `evalf` controls the number of floating-point digits for that particular calculation, and the special variable `Digits` sets the number of floating-point digits for all subsequent calculations.

```
> Digits := 20;
```

$$Digits := 20$$

```
> sin(0.2);
```

$$0.19866933079506121546$$

`Digits` is now set to twenty, which Maple then uses at each step in a calculation. Maple works like a calculator or an ordinary computer application in this respect. When you evaluate a complicated numerical expression, errors can accumulate to reduce the accuracy of the result to less than twenty digits. In general, setting `Digits` to produce a given accuracy is not easy, as the final result depends on your particular question. Using larger values, however, usually gives you some indication. If required, Maple can provide extreme floating-point accuracy.

Arithmetic with Special Numbers

Maple can work with complex numbers. I is the Maple default symbol for the square root of minus one, that is, $I = \sqrt{-1}$.

```
> (2 + 5*I) + (1 - I);
```

$$3 + 4\,I$$

```
> (1 + I)/(3 - 2*I);
```

$$\frac{1}{13} + \frac{5}{13}\,I$$

You can also work with other *bases and number systems.*

```
> convert(247, binary);
```

$$11110111$$

```
> convert(1023, hex);
```

$$\mathit{3FF}$$

```
> convert(17, base, 3);
```

$$[2,\ 2,\ 1]$$

Maple returns an integer base conversion as a list of digits; otherwise, a line of numbers, like 221, may be ambiguous, especially when dealing with large bases. Note that Maple lists the digits in order from least significant to most significant.

Maple also supports arithmetic in *finite rings and fields.*

```
> 27 mod 4;
```

$$3$$

Symmetric and *positive representations* are both available.

```
> mods(27,4);
```

$$-1$$

```
> modp(27,4);
```

$$3$$

The default for the `mod` command is positive representation, but you can change this option. For details, refer to `?mod`.

Maple can work with *Gaussian Integers*. The `GaussInt` package has about thirty commands for working with these special numbers. For information about these commands, refer to `?GaussInt` help page.

Mathematical Functions

Maple contains all the standard mathematical functions (see Table 2.2 for a partial list).

```
> sin(Pi/4);
```

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

```
> ln(1);
```

$$0$$

Table 2.2 Select Mathematical Functions in Maple

Function	***Description***
`sin`, `cos`, `tan`, etc.	trigonometric functions
`sinh`, `cosh`, `tanh`, etc.	hyperbolic trigonometric functions
`arcsin`, `arccos`, `arctan`, etc.	inverse trigonometric functions
`exp`	exponential function
`ln`	natural logarithmic function
`log[10]`	logarithmic function base 10
`sqrt`	algebraic square root function
`round`	round to the nearest integer
`trunc`	truncate to the integer part
`frac`	fractional part
`BesselI`, `BesselJ`, `BesselK`, `BesselY`	Bessel functions
`binomial`	binomial function
`erf`, `erfc`	error & complementary error functions
`Heaviside`	Heaviside step function
`Dirac`	Dirac delta function
`MeijerG`	Meijer G function
`Zeta`	Riemann Zeta function
`LegendreKc`, `LegendreKc1`, `LegendreEc`, `LegendreEc1`, `LegendrePic`, `LegendrePic1`	Legendre's elliptic integrals
`hypergeom`	hypergeometric function

Note: When Maple cannot find a simpler form, it leaves the expression as it is rather than convert it to an inexact form.

```
> ln(Pi);
```

$$\ln(\pi)$$

2.3 Basic Symbolic Computations

Maple can work with mathematical unknowns, and expressions which contain them.

```
> (1 + x)^2;
```

$$(1+x)^2$$

```
> (1 + x) + (3 - 2*x);
```

$$4-x$$

Note that Maple automatically simplifies the second expression.

Maple has hundreds of commands for working with symbolic expressions. For a partial list, see Table 2.2.

```
> expand((1 + x)^2);
```

$$1+2\,x+x^2$$

```
> factor(%);
```

$$(1+x)^2$$

As mentioned in **2.2 Numerical Computations**, the ditto operator, %, is a shorthand notation for the previous result.

```
> Diff(sin(x), x);
```

$$\frac{d}{dx}\sin(x)$$

```
> value(%);
```

$$\cos(x)$$

```
> Sum(n^2, n);
```

$$\sum_{n} n^2$$

```
> value(%);
```

$$\frac{1}{3}\,n^3-\frac{1}{2}\,n^2+\frac{1}{6}\,n$$

Divide one polynomial in x by another.

```
> rem(x^3+x+1, x^2+x+1, x);
```

$$2 + x$$

Create a series.

```
> series(sin(x), x=0, 10);
```

$$x - \frac{1}{6}\,x^3 + \frac{1}{120}\,x^5 - \frac{1}{5040}\,x^7 + \frac{1}{362880}\,x^9 + \mathrm{O}(x^{10})$$

All the mathematical functions mentioned in the previous section also accept unknowns as arguments.

2.4 Assigning Expressions to Names

This section introduces the following concepts in Maple.

- Naming an object
- Guidelines for Maple names
- Maple arrow notation (`->`)
- Assignment operator (`:=`)
- Predefined and reserved names

Syntax for Naming an Object

Using the ditto operator, or retyping a Maple expression every time you want to use it, is not always convenient, so Maple enables you to name an object. Use the following syntax for naming.

```
name := expression;
```

You can assign *any* Maple expression to a name.

```
> var := x;
```

$$var := x$$

```
> term := x*y;
```

$$term := x\,y$$

You can assign equations to names.

```
> eqn := x = y + 2;
```

$$eqn := x = y + 2$$

Guidelines for Maple Names

Maple names can include any alphanumeric characters and underscores, but they *cannot start with a number*. Do not start names with an underscore because Maple uses these names for internal classification.

- Examples of valid Maple names are `polynomial`, `test_data`, `RoOt_lOcUs_pLoT`, and `value2`.
- Examples of *invalid* Maple names are `2ndphase` (because it begins with a number) and `x&y` (because `&` is not an alphanumeric character).

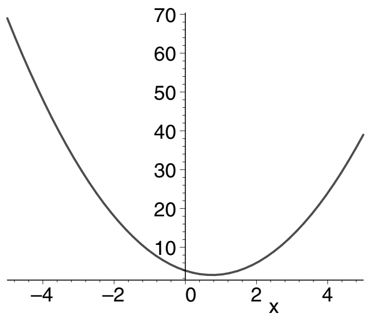
Maple Arrow Notation in Defining Functions

Define functions by using the Maple *arrow notation* (`->`). This notation allows you to evaluate a function when it appears in Maple expressions. You can do simple graphing of the function by using the `plot` command.

```
> f := x -> 2*x^2 -3*x +4;
```

$$f := x \to 2\,x^2 - 3\,x + 4$$

```
> plot (f(x), x= -5..5);
```



For more information on the `plot` command, see chapter 5 or enter `?plot` at the Maple prompt.

The Assignment Operator

The assignment (`:=`) operator associates a function name with a function definition. The name of the function is on the left-hand side of the `:=`. The function definition (using the arrow notation) is on the right-hand side. The following statement defines `f` as the *squaring function.*

```
> f := x -> x^2;
```

$$f := x \to x^2$$

Evaluating `f` at an argument produces the square of the argument of `f`.

```
> f(5);
```

$$25$$

```
> f(y+1);
```

$$(y + 1)^2$$

Predefined and Reserved Names

Maple has some predefined and reserved names. If you try to assign to a name that is predefined or reserved, Maple displays a message, informing you that the name you have chosen is protected.

```
> Pi := 3.14;

Error, attempting to assign to ‘Pi‘ which is protected
```

```
> set := {1, 2, 3};

Error, attempting to assign to ‘set‘ which is protected
```

2.5 Basic Types of Maple Objects

This section examines basic types of Maple objects, including *expression sequences*, *lists*, *sets*, *arrays*, *tables*, and *strings*. These ideas are essential to the discussion in the rest of this book. Also, the following concepts in Maple are introduced.

- Concatenation operator
- Square bracket usage
- Curly braces usage
- Mapping
- Colon (`:`) for suppressing output
- Double quotation mark usage

Types Expressions belong to a class or group that share common properities. The classes and groups are known as *types*. For a complete list of types in Maple, refer to the `?type` help page.

Expression Sequences

The basic Maple data structure is the *expression sequence*. This is a group of Maple expressions separated by commas.

```
> 1, 2, 3, 4;
```

$$1, 2, 3, 4$$

```
> x, y, z, w;
```

$$x, y, z, w$$

Expression sequences are neither lists nor sets. They are a distinct data structure within Maple and have their own properties.

- Expression sequences preserve the order and repetition of their elements. Items stay in the order in which you enter them. If you enter an element twice, both copies remain.
- Sequences are often used to build more sophisticated objects through such operations as concatenation.

Other properties of sequences will become apparent as you progress through this manual. Sequences extend the capabilities of many basic Maple operations. For example, concatenation is a basic name-forming operation. The *concatenation operator* in Maple is "`||`". You can use the operator in the following manner.

```
> a||b;
```

$$ab$$

When applying concatenation to a sequence, the operation affects each element. For example, if S is a sequence, then you can prepend the name `a` to each element in S by concatenating `a` and S.

```
> S := 1, 2, 3, 4;
```

$$S := 1,\ 2,\ 3,\ 4$$

```
> a||S;
```

$$\mathit{a1},\ \mathit{a2},\ \mathit{a3},\ \mathit{a4}$$

You can also perform multiple assignments using expression sequences. For example

```
> f,g,h := 3, 6, 1;
```

$$f,\ g,\ h := 3,\ 6,\ 1$$

```
> f;
```

$$3$$

```
> h;
```

$$1$$

Lists

You create a *list* by enclosing any number of Maple objects (separated by commas) in *square brackets.*

```
> data_list := [1, 2, 3, 4, 5];
```

$$data\_list := [1, 2, 3, 4, 5]$$

```
> polynomials := [x^2+3, x^2+3*x-1, 2*x];
```

$$polynomials := [x^2+3,\ x^2+3\,x-1,\ 2\,x]$$

```
> participants := [Kathy, Frank, Rene, Niklaus, Liz];
```

$$participants := [Kathy,\ Frank,\ Rene,\ Niklaus,\ Liz]$$

Thus, a list is an expression sequence enclosed in square brackets.

Order Maple preserves the order and repetition of elements in a list. Thus, `[a,b,c]`, `[b,c,a]`, and `[a,a,b,c,a]` are all different.

```
> [a,b,c], [b,c,a], [a,a,b,c,a];
```

$$[a,\ b,\ c],\ [b,\ c,\ a],\ [a,\ a,\ b,\ c,\ a]$$

Because order is preserved, you can extract a particular element from a list without searching for it.

```
> letters := [a,b,c];
```

$$letters := [a,\ b,\ c]$$

```
> letters[2];
```

$$b$$

Use the `nops` command to determine the number of elements in a list.

```
> nops(letters);
```

$$3$$

Section **2.6 Expression Manipulation** discusses this command, including its other uses, in more detail.

Sets

Maple supports *sets* in the mathematical sense. Commas separate the objects, as they do in a sequence or list, but the enclosing *curly braces* identify the object as a set.

```
> data_set := {1, -1, 0, 10, 2};
```

$$data\_set := \{-1, 0, 1, 2, 10\}$$

```
> unknowns := {x, y, z};
```

$$unknowns := \{x, y, z\}$$

Thus, a set is an expression sequence enclosed in curly braces.

Order Maple does *not* preserve order or repetition in a set. That is, Maple sets have the same properties as sets do in mathematics. Thus, the following three sets are identical.

```
> {a,b,c}, {c,b,a}, {a,a,b,c,a};
```

$$\{a, b, c\}, \{a, b, c\}, \{a, b, c\}$$

For Maple, the integer 2 is distinct from the floating-point approximation 2.0. Thus, the following set has three elements, not two.

```
> {1, 2, 2.0};
```

$$\{1, 2, 2.0\}$$

The properties of sets make them a particularly useful concept in Maple, just as they are in mathematics. Maple provides many operations on sets, including the basic operations of *intersection* and *union* using the notation `intersect` and `union`.

```
> {a,b,c} union {c,d,e};
```

$$\{a, b, c, d, e\}$$

```
> {1,2,3,a,b,c} intersect {0,1,y,a};
```

$$\{1, a\}$$

The `nops` command counts the number of elements in a set or list.

```
> nops(%);
```

$$2$$

For more details on the `nops` command, see **2.6 Expression Manipulation**.

Mapping A common and useful command, often used on sets, is `map`. Mapping applies a function simultaneously to all the elements of any structure.

```
> numbers := {0, Pi/3, Pi/2, Pi};
```

$$numbers := \{0, \pi, \frac{1}{3}\pi, \frac{1}{2}\pi\}$$

```
> map(g, numbers);
```

$$\{\mathrm{g}(0), \mathrm{g}(\pi), \mathrm{g}(\frac{1}{3}\pi), \mathrm{g}(\frac{1}{2}\pi)\}$$

```
> map(sin, numbers);
```

$$\{0, 1, \frac{1}{2}\sqrt{3}\}$$

Further examples demonstrating the use of `map` appear in **2.6 Expression Manipulation** and **6.3 Structural Manipulations**.

Operations on Sets and Lists

The `member` command verifies membership in sets and lists.

```
> participants := [Kate, Tom, Steve];
```

$$participants := [Kate, Tom, Steve]$$

```
> member(Tom, participants);
```

$$true$$

```
> data_set := {5, 6, 3, 7};
```

$$data\_set := \{3, 5, 6, 7\}$$

```
> member(2, data_set);
```

$$false$$

To select items from lists, use the subscript notation, `[n]`, where n identifies the position of the desired element in the list.

```
> participants[2];
```

$$Tom$$

Maple recognizes *empty* sets and lists, that is, lists or sets that have no elements.

```
> empty_set := {};
```

$$empty\_set := \{\}$$

```
> empty_list := [];
```

$$empty\_list := []$$

You can create a new set from other sets by using, for example, the `union` command. Delete items from sets by using the `minus` command.

```
> old_set := {2, 3, 4} union {};
```

$$old\_set := \{2, 3, 4\}$$

```
> new_set := old_set union {2, 5};
```

$$new\_set := \{2, 3, 4, 5\}$$

```
> third_set := old_set minus {2, 5};
```

$$third\_set := \{3, 4\}$$

Arrays

Arrays are an extension of the concept of the list data structure. Think of a list as a group of items in which you associate each item with a positive integer, its index, that represents its position in the list. The Maple `array` data structure is a generalization of this idea. Each element is still associated with an index, but an array is not restricted to one dimension. In addition, indices can also be zero or negative. Furthermore, you can define or change the array's individual elements without redefining it entirely.

Declare the array to indicate dimensions.

```
> squares := array(1..3);
```

$$squares := \mathrm{array}(1..3, [])$$

Assign the array elements. Multiple commands can be entered at one command prompt provided each ends with a colon or semicolon.

```
> squares[1] := 1;  squares[2] := 2^2;  squares[3] := 3^2;
```

$$squares_1 := 1$$

$$squares_2 := 4$$

$$squares_3 := 9$$

Or do both simultaneously.

```
> cubes := array( 1..3, [1,8,27] );
```

$$cubes := [1, 8, 27]$$

You can select a single element using the same notation applied to lists.

```
> squares[2];
```

$$4$$

You must declare arrays in advance. To see the array's contents, you must use a command such as `print`.

```
> squares;
```

$$squares$$

```
> print(squares);
```

$$[1, 4, 9]$$

The preceding array has only one dimension, but arrays can have more than one dimension. Define a 3×3 array.

```
> pwrs := array(1..3,1..3);
```

$$pwrs := \mathrm{array}(1..3, 1..3, [])$$

This array has dimension two (two sets of indices). To begin, assign the array elements of the first row.

```
> pwrs[1,1] := 1;  pwrs[1,2] := 1;  pwrs[1,3] := 1;
```

$$pwrs_{1,1} := 1$$

$$pwrs_{1,2} := 1$$

$$pwrs_{1,3} := 1$$

Continue for the rest of the array. If you prefer, you can end each command with a colon (`:`), instead of the usual semicolon (`;`), to *suppress the output*. Both the colon and semicolon are statement separators.

```
> pwrs[2,1] := 2:  pwrs[2,2] := 4:  pwrs[2,3] := 8:
> pwrs[3,1] := 3:  pwrs[3,2] := 9:  pwrs[3,3] := 27:
> print(pwrs);
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \end{bmatrix}$$

You can select an element by specifying both the row and column.

```
> pwrs[2,3];
```

$$8$$

You can define a two-dimensional array and its elements simultaneously by using a similar method employed for the one-dimensional example shown earlier. To do so, use lists within lists. That is, make a list where each element is a list that contains the elements of one row of the array. Thus, you could define the `pwrs` array as follows.

```
> pwrs2 := array( 1..3, 1..3, [[1,1,1], [2,4,8], [3,9,27]] );
```

$$pwrs2 := \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \end{bmatrix}$$

Arrays are not limited to two dimensions, but those of higher order are more difficult to display. You can declare all the elements of the array as you define its dimension.

```
> array3 := array( 1..2, 1..2, 1..2,
> [[[1,2],[3,4]], [[5,6],[7,8]]] );
```

$$\begin{aligned}
&array3 := \mathrm{array}(1..2,\ 1..2,\ 1..2,\ [\\
&(1,\ 1,\ 1) = 1\\
&(1,\ 1,\ 2) = 2\\
&(1,\ 2,\ 1) = 3\\
&(1,\ 2,\ 2) = 4\\
&(2,\ 1,\ 1) = 5\\
&(2,\ 1,\ 2) = 6\\
&(2,\ 2,\ 1) = 7\\
&(2,\ 2,\ 2) = 8\\
&])
\end{aligned}$$

Maple does not automatically expand the name of an array to the representation of all elements. In some commands, you must specify explicitly that you want to perform an operation on the elements.

Suppose that you want to define a new array identical to `pwr`, but with each occurrence of the number 2 in `pwrs` replaced by the number 9. To perform this substitution, use the `subs` command. The basic syntax is

```
subs( x=expr1, y=expr2, ... , main_expr )
```

Note: The `subs` command does not modify the value of `main_expr`. It returns an object of the same type with the specified substitutions. For example, to substitute $x + y$ for z in an expression, do the following.

```
> expr := z^2 + 3;
```

$$expr := z^2 + 3$$

```
> subs( {z=x+y}, expr);
```

$$(x + y)^2 + 3$$

Note that the following call to `subs` does not work.

```
> subs( {2=9}, pwrs );
```

$$pwrs$$

You must instead force Maple to fully evaluate the name of the array to the component level and not just to its name, using the command `evalm`.

```
> pwrs3:=subs( {2=9}, evalm(pwrs) );
```

$$pwrs3 := \begin{bmatrix} 1 & 1 & 1 \\ 9 & 4 & 8 \\ 3 & 9 & 27 \end{bmatrix}$$

This causes the substitution to occur in the components and full evaluation displays the array's elements, similar to using the `print` command.

```
> evalm(pwrs3);
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 9 & 4 & 8 \\ 3 & 9 & 27 \end{bmatrix}$$

Tables

A *table* is an extension of the concept of the array data structure. The difference between an array and a table is that a table can have *anything* for indices, not just integers.

```
> translate := table([one=un,two=deux,three=trois]);
```

$$translate := \mathrm{table}([two = deux,\ three = trois,\ one = un])$$

```
> translate[two];
```

$$deux$$

Although at first they may seem to have little advantage over arrays, table structures are very powerful. Tables enable you to work with natural notation for data structures. For example, you can display the physical properties of materials using a Maple table.

```
> earth_data := table( [ mass=[5.976*10^24,kg],
>                        radius=[6.378164*10^6,m],
>                        circumference=[4.00752*10^7,m] ] );
```

$$\begin{aligned}&earth\_data := \mathrm{table}([mass = [0.5976000000\,10^{25},\ kg],\\&radius = [0.6378164000\,10^{7},\ m],\\&circumference = [0.4007520000\,10^{8},\ m]\\&])\end{aligned}$$

```
> earth_data[mass];
```

$$[0.5976000000\,10^{25},\ kg]$$

In this example, each index is a name and each entry is a list. Often, much more general indices are useful. For example, you could construct a table which has algebraic formulæ for indices and the derivatives of these formulæ as values.

Strings

A *string* is also an object in Maple and is created by enclosing any number of characters in *double quotes*.

```
> "This is a string.";
```

$$\text{“This is a string.”}$$

They are nearly indivisible constructs that stand only for themselves; they cannot be assigned a value.

```
> "my age" := 32;

Error, invalid left hand side of assignment
```

Like elements of lists or arrays, the individual characters of a string can be indexed with square bracket notation.

```
> mystr := "I ate the whole thing.";
```

$$mystr := \text{“I ate the whole thing.”}$$

```
> mystr[3..5];
```

$$\text{“ate”}$$

```
> mystr[11..-2];
```

$$\text{“whole thing”}$$

A negative index represents a character position counted from the right end of the string. In the example above, -2 represents the second last character.

The concatenation operator, “`||`”, or the `cat` command is used to join two strings together, and the `length` command is used to determine the number of characters in a string.

```
> newstr := cat("I can’t believe ", mystr);
```

$$newstr := \text{“I can’t believe I ate the whole thing.”}$$

```
> length(newstr);
```

For examples of commands that operate on strings and take strings as input, refer to the `?StringTools` help page.

2.6 Expression Manipulation

Many Maple commands concentrate on manipulating expressions. This includes manipulating results of Maple commands into a familiar or useful form. This section introduces the most commonly used commands in this area.

The `simplify` Command

You can use this command to apply simplification rules to an expression. Maple has simplification rules for various types of expressions and forms, including trigonometric functions, radicals, logarithmic functions, exponential functions, powers, and various special functions.

```
> expr := cos(x)^5 + sin(x)^4 + 2*cos(x)^2
> - 2*sin(x)^2 - cos(2*x);
```

$$expr := \cos(x)^5 + \sin(x)^4 + 2\cos(x)^2 - 2\sin(x)^2 - \cos(2\,x)$$

```
> simplify(expr);
```

$$\cos(x)^4\,(\cos(x) + 1)$$

To perform only a certain type of simplification, specify the type you want.

```
> simplify(sin(x)^2 + ln(2*y) + cos(x)^2);
```

$$1 + \ln(2) + \ln(y)$$

```
> simplify(sin(x)^2 + ln(2*y) + cos(x)^2, 'trig');
```

$$1 + \ln(2\,y)$$

```
> simplify(sin(x)^2 + ln(2*y) + cos(x)^2, 'ln');
```

$$\sin(x)^2 + \ln(2) + \ln(y) + \cos(x)^2$$

With the *side relations* feature, you can apply your own simplification rules.

```
> siderel := {sin(x)^2 + cos(x)^2 = 1};
```

$$siderel := \{\sin(x)^2 + \cos(x)^2 = 1\}$$

```
> trig_expr := sin(x)^3 - sin(x)*cos(x)^2 + 3*cos(x)^3;
```

$$trig\_expr := \sin(x)^3 - \sin(x)\cos(x)^2 + 3\cos(x)^3$$

```
> simplify(trig_expr, siderel);
```

$$2\sin(x)^3 - 3\cos(x)\sin(x)^2 + 3\cos(x) - \sin(x)$$

The `factor` Command

This command factors polynomial expressions.

```
> big_poly := x^5 - x^4 - 7*x^3 + x^2 + 6*x;
```

$$big\_poly := x^5 - x^4 - 7x^3 + x^2 + 6x$$

```
> factor(big_poly);
```

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

```
> rat_expr := (x^3 - y^3)/(x^4 - y^4);
```

$$rat\_expr := \frac{x^3 - y^3}{x^4 - y^4}$$

Both the numerator and denominator contain the common factor $x-y$. Thus, factoring cancels these terms.

```
> factor(rat_expr);
```

$$\frac{x^2 + x\,y + y^2}{(y + x)\,(x^2 + y^2)}$$

Maple can factor both univariate and multivariate polynomials over the domain the coefficients specify. You can also factor polynomials over algebraic extensions. For details, refer to the `?factor` help page.

The `expand` Command

The `expand` command is essentially the reverse of `factor`. It causes the expansion of multiplied terms as well as a number of other expansions. This is among the most useful of the manipulation commands. Although you might imagine that with a name like `expand` the result would be larger and more complex than the original expression; this is not always the case. In fact, expanding some expressions results in substantial simplification.

```
> expand((x+1)*(x+2));
```

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

```
> expand(sin(x+y));
```

$$\sin(y)\cos(x) + \cos(y)\sin(x)$$

```
> expand(exp(a+ln(b)));
```

$$e^a\,b$$

The `expand` command is quite flexible. You can you specify that certain subexpressions be unchanged by the expansion and program custom expansion rules.

Although the `simplify` command may seem to be the most useful command, this is misleading. Unfortunately, the word *simplify* is rather vague. When you request to `simplify` an expression, Maple examines your expression, tests many techniques, and then tries applying the appropriate simplification rules. However, this might take a little time. As well, Maple may not be able to determine what you want to accomplish since universal mathematical rules do not define what is simpler.

When you do know which manipulations will make your expression simpler for you, specify them directly. In particular, the `expand` command is among the most useful. It frequently results in substantial simplification, and also leaves expressions in a convenient form for many other commands.

The `convert` Command

This command converts expressions between different forms. For a list of common conversions, see Table 2.3.

```
> convert(cos(x),exp);
```

$$\frac{1}{2}e^{(x\,I)}+\frac{1}{2}\frac{1}{e^{(x\,I)}}$$

```
> convert(1/2*exp(x) + 1/2*exp(-x),trig);
```

$$\cosh(x)$$

```
> A := Matrix([[a,b],[c,d]]);
```

$$A := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> convert(A, 'listlist');
```

$$[[a, b], [c, d]]$$

```
> convert(A, 'set');
```

$$\{a, b, d, c\}$$

```
> convert(%, 'list');
```

$$[a, b, d, c]$$

The `normal` Command

This command transforms rational expressions into *factored normal form*,

$$\frac{numerator}{denominator},$$

Table 2.3 Common Conversions

Argument	***Description***
`polynom`	series to polynomials
`exp`, `expln`, `expsincos`	trigonometric expressions to exponential form
`parfrac`	rational expressions to partial fraction form
`rational`	floating-point numbers to rational form
`radians`, `degrees`	between degrees and radians
`set`, `list`, `listlist`	between data structures
`temperature`	between temperature scales
`units`	between units

where the *numerator* and *denominator* are relatively prime polynomials with integer coefficients.

```
> rat_expr_2 := (x^2 - y^2)/(x - y)^3 ;
```

$$rat\_expr\_2 := \frac{x^2 - y^2}{(-y+x)^3}$$

```
> normal(rat_expr_2);
```

$$\frac{y+x}{(-y+x)^2}$$

```
> normal(rat_expr_2, 'expanded');
```

$$\frac{y+x}{y^2 - 2\,x\,y + x^2}$$

The `expanded` option transforms rational expressions into *expanded normal form*.

The `combine` Command

This command combines terms in sums, products, and powers into a single term. These transformations are, in some cases, the reverse of the transformations that `expand` applies.

```
> combine(exp(x)^2*exp(y),exp);
```

$$e^{(2\,x+y)}$$

```
> combine((x^a)^2, power);
```

$$x^{(2\,a)}$$

The `map` Command

This command is useful when working with lists, sets, or arrays. It provides a means for working with multiple solutions or for applying an operation to each element of an array.

The `map` command applies a command to each element of a data structure or expression. While it is possible to write program structures such as loops to accomplish these tasks, you should not underestimate the convenience and power of the `map` command. The `map` command is one of the most useful commands in Maple.

```
> map( f, [a,b,c] );
```

$$[\mathrm{f}(a),\, \mathrm{f}(b),\, \mathrm{f}(c)]$$

```
> data_list := [0, Pi/2, 3*Pi/2, 2*Pi];
```

$$data\_list := [0,\, \frac{1}{2}\,\pi,\, \frac{3}{2}\,\pi,\, 2\,\pi]$$

```
> map(sin, data_list);
```

$$[0,\, 1,\, -1,\, 0]$$

If you give the `map` command more than two arguments, Maple passes the last argument(s) to the initial command.

```
> map( f, [a,b,c], x, y );
```

$$[\mathrm{f}(a,\, x,\, y),\, \mathrm{f}(b,\, x,\, y),\, \mathrm{f}(c,\, x,\, y)]$$

For example, to differentiate each item in a list with respect to x, you can use the following commands.

```
> fcn_list := [sin(x),ln(x),x^2];
```

$$fcn\_list := [\sin(x),\, \ln(x),\, x^2]$$

```
> map(Diff, fcn_list, x);
```

$$[\frac{d}{dx}\sin(x),\ \frac{d}{dx}\ln(x),\ \frac{d}{dx}(x^2)]$$

```
> map(value, %);
```

$$[\cos(x),\ \frac{1}{x},\ 2\,x]$$

You can also create an operation to map onto a list. For example, suppose that you want to square each element of a list. Replace each element (represented by x) with its square (x^2).

```
> map(x->x^2, [-1,0,1,2,3]);
```

$$[1,\ 0,\ 1,\ 4,\ 9]$$

The `lhs` and `rhs` Commands

These two commands take the left-hand side and right-hand side of an expression, respectively.

```
> eqn1 := x+y=z+3;
```

$$\mathit{eqn1} := y + x = z + 3$$

```
> lhs(eqn1);
```

$$y + x$$

```
> rhs(eqn1);
```

$$z + 3$$

The `numer` and `denom` Commands

These two commands take the numerator and denominator of a rational expression, respectively.

```
> numer(3/4);
```

$$3$$

```
> denom(1/(1 + x));
```

$$x + 1$$

The `nops` and `op` Commands

These two commands are useful for breaking expressions into parts and extracting subexpressions.

The `nops` command returns the number of parts in an expression.

```
> nops(x^2);
```

$$2$$

```
> nops(x + y + z);
```

$$3$$

The `op` command allows you to access the parts of an expression. It returns the parts as a sequence.

```
> op(x^2);
```

$$x, 2$$

You can also specify items by number or range.

```
> op(1, x^2);
```

$$x$$

```
> op(2, x^2);
```

$$2$$

```
> op(2..-2, x+y+z+w);
```

$$y, z$$

Common Questions about Expression Manipulation

1. How do I substitute for a product of two unknowns? Use side relations to specify an identity. Substituting directly does not usually work because Maple searches for an exact match before substituting.

```
> expr := a^3*b^2;
```

$$expr := a^3\, b^2$$

```
> subs(a*b=5, expr);
```

$$a^3\, b^2$$

The `subs` command was unsuccessful in its attempt to substitute. Use the `simplify` command.

```
> simplify(expr, {a*b=5});
```

$$25\, a$$

You can also use the `algsubs` command, which performs an algebraic substitution.

```
> algsubs(a*b=5, expr);
```

$$25\, a$$

2. How do I factor out the constant from $2x + 2y$? Currently, this operation is not possible in Maple because its simplifier automatically distributes the number over the product, believing that a sum is simpler than a product. In most cases, this is true.

If you enter the following expression, Maple automatically multiplies the constant into the expression.

```
> 2*(x + y);
```

$$2\, x + 2\, y$$

How can you then deal with such expressions, when you need to factor out constants, or negative signs? To factor such expressions, try this substitution.

```
> expr3 := 2*(x + y);
```

$$expr3 := 2\,x + 2\,y$$

```
> subs( 2=two, expr3 );
```

$$x\,two + y\,two$$

```
> factor(%);
```

$$two\,(x + y)$$

2.7 Conclusion

In this chapter you have seen many of the types of objects which Maple is capable of manipulating, including sequences, sets, and lists. You have seen a number of commands, including `expand`, `factor`, and `simplify`, that are useful for manipulating and simplifying algebraic expressions. Others, such as `map`, are useful for sets, lists, and arrays. Meanwhile, `subs` is useful almost any time.

In the next chapter, you will learn to apply these concepts to solve systems of equations, one of the most fundamental problems in mathematics. As you learn about new commands, observe how the concepts of this chapter are used in setting up problems and manipulating solutions.