Maple for Math Majors

Roger Kraft
Department of Mathematics, Computer Science, and Statistics
Purdue University Calumet
roger@calumet.purdue.edu

1. Maple Basics

This worksheet helps you get started using Maple and its user interface. It also introduces you to some of Maple's basic capabilities. This worksheet contains five sections. Click on a plus sign below to expand a section.

1.1. Introduction

The main purpose of this worksheet is to introduce you to Maple. As you will soon see, this worksheet is an interactive document. You will do more than just read it, you will work with it. You can make changes to it and try variations on what is in it. In other words, as you work through this worksheet you will not just be reading about Maple, you will actually be doing something with Maple.

Either click on the plus sign below or, if the cursor is on the next prompt, just hit the Enter key to open the next section of this worksheet and begin working.

```
>
```

1.2. Getting started with Maple

A Maple worksheet is made up of three components, Maple commands (in red), the output (in blue) that Maple produces for each command, and explanations (in black). Here is an example of a Maple command and its output.

```
> 1+1;
```

$$2$$

Some of the commands in this worksheet, like the one just above, have already been executed and their output is right below the command. But most of the commands in this worksheet have not yet been executed and so you need to tell Maple to execute them so that you can see the results. Here is an example of a Maple command that has not been executed yet. To execute this command, click on it with the mouse and then hit the Enter key.

```
> 2+2;
```

Notice how Maple produced the output and then the cursor jumped down to the next Maple command, skipping over this explanation. From now on, this is pretty much how things will go in this worksheet; the cursor is on a Maple command, you hit the Enter key, Maple executes the command and displays the result, the cursor jumps down to the next Maple command, and you read the explanation (if any) that is between the commands. (Occasionally, this skipping over the explanation between commands can cause Maple to scroll what you want to read right off of the top of the screen. When that happens, you need to use the vertical scroll bar on the edge of the window to scroll back down the screen.)

Every Maple command is next to a prompt (the greater than sign >), and every Maple command must be terminated by a semicolon or colon. The semicolon tells Maple to print the result of the command, the colon tells Maple not to print the command's result. Try executing the next command.

```
> 3+3:
```

A Maple command can contain a comment that is not really part of the command. A pound sign (i.e. #) as part of a line of Maple input means a comment that Maple should ignore.

```
> 4+4;  # Maple ignores what comes after the pound sign.
```

Here is an empty Maple prompt. When the cursor is on an empty Maple prompt and you hit the Enter key, the cursor will jump down to the next prompt (try it).

```
>
```

Here is another empty Maple prompt. Type in a simple Maple command and execute it.

```
>
```

You can go back and edit Maple commands that have already been executed. Click on the Maple command that you just executed and use the arrow keys and backspace key to move around the command and change it. Then hit the Enter key to execute your new command.

If you make some kind of mistake when you type in a Maple command, and Maple cannot understand what you want it to do when you execute the command, then Maple gives you some kind of an error message as the output. For example, a common mistake is to forget a semicolon (or colon) at the end of a Maple command. Try executing the following command.

```
> 5+5
```

If you have forgotten a semicolon, just go back to the command and put a semicolon at the end and re-execute the command. Try that with the last command.

There are many other kinds of mistakes that you can make in a Maple command. One of the hardest things about using any kind of computer program is dealing with mistakes and the vague error messages that go along with them.

```
> 6+;
```

It takes a fair amount of experience before you will know how to deal with most of Maple's error messages, and you can be sure that you will see a lot of them. In the mean time, type your commands very carefully to avoid as many typos as you can. (Fix the above command so that it no longer causes an error message.)

If you want to, you can put more than one Maple command on a line but each of the Maple commands must be terminated with its own colon or semicolon. The following three commands will all execute at the same time.

```
> 7+7; 8+8; 9+9;
```

(Try deleting one of the semicolons and then re-executing the line, or replace a semicolon with a colon or a comma.)

There is another way to get several Maple commands to execute at the same time, and that is to put the commands in an **execution group**. The following three Maple commands all execute together; you can place the cursor anywhere within this execution group and when you press the Enter key, all three commands are executed.

```
> 10+10;
> 11+11;
> 12+12;
```

Notice that it is the bracket just to left of the prompts that tells you the three commands are tied together into an execution group (if you do not see the bracket, use the "View -> Show Group ranges" menu item to turn them on). Execution groups can be a bit cumbersome to work with. There is more about them later in this worksheet. For now, if you want to execute several simple Maple commands at the same time, it is much easier to put the commands on a single line than it is to create an execution group.

So much for the really basic basics of using Maple. Let us go on and see what makes Maple so useful. If the cursor is at the next prompt, just hit the Enter key and Maple will automatically jump to the first prompt in the next section of this worksheet.

```
>
```

1.3. Basic Maple commands

The following four subsections contain a lot of examples of using Maple's basic commands. In these subsections you will find many empty Maple prompts. Take these as an invitation to modify the examples that are given and to try out your own examples. You can even add additional Maple prompts to this worksheet by placing the cursor anywhere and typing Ctrl-J. (Place the cursor at the end of this paragraph and try it right now.) Do not be afraid to "mess up" this worksheet. You can always download another copy of it.

If the cursor is at the next prompt, just hit the Enter key and Maple will automatically jump to the first prompt in the next subsection.

```
>
```

1.3.1. Maple is a numeric calculator.

Maple can be used much like a traditional calculator but it does a great deal more than typical numeric calculators. Maple can do both "symbolic arithmetic" and "numeric arithmetic". By symbolic arithmetic I mean arithmetic the way that you were taught to do it with paper and pencil. By numeric arithmetic I mean what a typical handheld calculator does. Here are some examples.

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

Notice that Maple did the addition of fractions symbolically. If we put a decimal point in the command, then Maple will do the arithmetic numerically, like a calculator.

```
> 1.0/3 + 1/3;  # Notice the decimal point.
```

Let us check that one third plus one third is two thirds. Here is Maple's decimal value for two thirds.

```
> 2/3.0;
```

Notice that, when computed numerically, one third plus one third was *not* the same as two thirds. This shows that there is a difference in Maple between symbolic and numeric calculations. We sometimes say that symbolic calculations are "exact" but numeric calculations are an "approximation". Both .6666666666 and .6666666667 are approximations of 2/3. Neither of them is "correct"; in fact they are both "wrong"! (In a certain sense though one of them is better than the other. Which one?) This is an idea that we will return to latter on.

```
>
```

Here are some other symbolic calculations.

```
> 18/27; 9/15;
```

Notice that Maple simplified the fractions.

```
> 2/3 + 3/5;
```

Notice that Maple put the fractions over a common denominator. If we want to know the decimal value of this last result, there are two ways to get it. First, we can put a decimal point in the input.

```
> 2/3. + 3/5;
```

The other way is to use the Maple command **evalf**, which converts a number to its decimal representation.

```
> evalf( 2/3 + 3/5 );
```

eval is short for evaluate and the **f** is short for "floating-point" which is computerese for a decimal number. So **evalf** means "**eval**uate to a **f**loating-point number".

```
>
```

One advantage of **evalf** is that you can request as many decimal places as you like. Here is an example with 100 decimal places.

```
> 2/7 + 3/5; evalf( 2/7 + 3/5, 100);
```

Notice that in the last example I had to type the expression **2/7 + 3/5** twice. There is a shorthand in Maple that helps eliminate this kind of redundancy. The percent symbol (**%**) in Maple is a shorthand for the "last result". So I can do the following.

```
> 2/7 + 3/5; evalf(%, 100);
```

In the second command, the percent sign represented 31/35, which was the result of the first command. There is even a shorthand for the "second to last result". So we can do the following.

```
> 90/35; 105/25; % + %%;
```

In the third command, the **%** represented 21/5 and the **%%** represented 18/7.

```
>
```

We can also give names to numbers. We can do the following, where **x** becomes a name for 237/35.

```
> x := 21/5 + 18/7;
```

The combination of a colon and an equal sign is call the **assignment operator** and it makes the name on the left hand side represent the value on the right hand side. Now we can use the name **x** anywhere we want in place of 237/35.

```
> evalf(x, 60);
>
```

Maple can do symbolic calculations with even more complicated expressions.

```
> sin( Pi/4 );
> tan( Pi/3 );
> arcsin( 1/2 );
```

But Maple cannot always come up with a symbolic answer.

```
> sin( Pi/30 );
```

When Maple cannot come up with a symbolic answer it returns an "unevaluated function call", that is, Maple pretty much repeats exactly what we typed in. In this case, we can use **evalf** to get a decimal approximation.

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

Sometimes, when you get an "unevaluated function call" as an answer, it simply means that you mistyped something and Maple does not know how to evaluated your input.

```
> sun( Pi/2 );
> evakf( Pi );
```

If you get an unevaluated function in your output, look carefully at your input for any typos before you try to do anything else.

```
>
```

Note: Maple actually can give us a symbolic result for $\sin\left(\frac{\pi}{30}\right)$. We just have to work a little harder.

```
> sin( Pi/30 );
> convert( %, radical );
>
```

Notice that **Pi** is Maple's way of representing the ratio of the circumference to the diameter of a circle. To Maple, **Pi** is a symbol for the exact value of this ratio. If you want a decimal approximation, then you have to convert the exact value to a decimal number.

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

(Notice that the last three commands were in an execution group so they were all calculated at the same time.)

Try asking for ten thousand decimal places of **Pi**.

```
>
```

If you really did ask for ten thousand decimal places of **Pi**, then the fastest way to get rid of them is to use the key combination Ctrl-z, which will undo the last result.

Maple can work with exponents also. The expression $25^{\left(\frac{1}{6}\right)}$ should simplify to $5^{\left(\frac{1}{3}\right)}$ (why?). Let us try it.

```
> 25^(1/6);
```

Maple did not do the simplification. Sometimes Maple needs to be explicitly told to do a simplification by using the **simplify** command.

```
> simplify( % );
```

Maple evaluates expressions using the usual algebraic precedence rules (if you do not put in parentheses). Here are two examples, with and without parentheses.

```
> 5-3/2+2*3;     (5-(3/2))+(2*3);
> 2*4^2/6+3*2;   ((2*(4^2))/6)+(3*2);
```

In the next two examples the parentheses change the order of operations from what they are when there are no parentheses. Compare these examples with the previous ones.

```
> (5-3)/(2+2*3);   5-(3/2+2*3);
> 2*4^2/(6+3)*2;   (2*4)^(2/6)+3*2;
>
```

Maple has built into it all the functions found on a scientific calculator plus a whole lot of other functions. This sentence contains a hyperlink to a list of Maple's built in functions. We can also define our own functions like this.

```
> f := t^2*sin(t);
```

Now we can evaluate our function using the **eval** command (**eval** is an abbreviation for **eval**uate).

```
> eval( f, t=Pi/4 );
```

We can also evaluate our function using the **subs** command (**subs** is an abbreviation of **subs**titute).

```
> subs( t=Pi/4, f );
```

Notice that, unlike the **eval** command, **subs** did not simplify the result so we also need to use **simplify**.

```
> simplify( % );
```

We can get a numeric approximation of this value.

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

Now we can plug a different number into our function.

```
> eval( f, t=8 );
```

Here is another example.

```
> f := exp(sqrt(t))*(1+arcsin(sec(t)));
> eval( f, t=ln(2) );
> evalf( % );
```

(Notice that the answer is a complex number. **I** is Maple's notation for $\sqrt{-1}$.)

```
>
```

Each of the following two commands will bring up a (long) list of all the functions that are built into Maple.

```
> ?inifcn
> ?index,functions
```

Maple can also calculate with integers in fairly sophisticated ways. Maple can compute factorials.

```
> 100!;   # This is read "100 factorial".
```

Maple can determine the number of digits in a large number.

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

Maple can find the prime factorization of any integer (this is actually a very difficult calculation to make).

```
> ifactor( %% );  # What does %% represent here?
```

Maple can test an integer and determine if it is a prime number or not.

```
> isprime( 2^19 - 1 );
```

Maple can find the greatest common divisor of two integers.

```
> igcd( 6!, 7! );
> igcd( 231, 260 );
```

Why is the last result explained by the next two results?

```
> ifactor( 231 );
> ifactor( 260 );
>
```

The next command will bring up a help page with a list of Maple commands for computing with integers.

```
> ?integers

>
```

1.3.2. Maple is a symbolic calculator.

Now we look at examples of what makes Maple such a useful tool for doing mathematics. Maple can do symbolic manipulations of expressions containing unknown variables. In other words, Maple can do all of high school algebra.

But before doing some algebra with Maple, we want to change **x** back into an "unknown variable." Earlier in this worksheet we gave **x** a value; now we need to take this value away somehow.

```
> x;     # x has a value.
```

We take away the value of **x** by using the assignment operator to assign an "unevaluated **x**" to **x** itself. We get an "unevaluated **x**" by putting right quotes on either side of the variable **x**, like this.

```
> x := 'x';  # Those are both right quotes.
```

This may look kind of mysterious but we will see exactly how it works in a later worksheet. But you need to remember how to "unassign" a variable. This is something that needs to be done quite often in Maple. Let us check that **x** really is unassigned now, that is, that it does not have a value.

```
> x;
```

Now that we have made **x** back into an "unknown variable", we can ask Maple to do some algebraic manipulations of equations in **x**.

The **solve** command is used to solve an equation for some unknown. You give the command the equation to solve and the variable to solve for. Here is an easy example.

```
> solve( 5*x+3=2, x );
```

Maple tells us what value of **x** solves the equation. You may wonder why we need to tell Maple to solve for **x** when in fact **x** is the only variable in the equation. But it may be that the equation has several variables, so we always need to tell Maple what variable to solve for. For example, the next two commands ask Maple to solve the same equation but for different variables.

```
> solve( a*x+b=c, x );
> solve( a*x+b=c, b );
```

Here is an example where we find the symbolic solution of an equation and then a numeric approximation of the solution. First find the symbolic solution.

```
> solve( x^2+2*x-1=0, x );
```

Notice that Maple found two solutions. Now compute numeric approximations.

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

The next command may seem quite strange to you at first but, as we will see later, it is an idea that you are already very familiar with. The next Maple command uses the assignment operator to give a name to an equation. The name of the equation will be **eqn1**.

```
> eqn1 := a*x^2+b*x+c=0;
```

Notice how there are two distinct kinds of "equal signs" in the above command, one is the assignment operator and the other is part of the equation. The first "equal sign" says that **eqn1** is a name for **a*x^2+b*x+c=0**. The second "equal sign" asks if the left hand side of the equation is equal to the right hand side.We can use the name to represent the equation in other Maple commands.

```
> solve( eqn1, x );
```

That answer should look familiar to you. Now let us give the unknowns **a**, **b**, and **c** values.

```
> a := 1; b := 2; c := -1;
```

Let us look at **eqn1** again.

```
> eqn1;
```

It is the same quadratic equation that we used a little while ago. Let us solve it again, but this time we will give the solution a name.

```
> solution := solve( eqn1, x );
```

The name **solution** now refers to both of the solutions! Here is how we can give a name to each individual solution.

```
> solution[1]; solution[2];
```

Suppose we want to check that these solutions really do solve the quadratic. We do that by "plugging in" each solution, one at a time, into the equation and checking that the left and right hand sides of the equation are the same. Here are Maple commands to do this. We use the **subs** (for "**subs**titute") command to plug the second solution back into the quadratic.

```
> subs( x=solution[2], eqn1 );
```

How do we know if the left and right hand sides are the same? Ask Maple to **simplify** the last result.

```
> simplify( % );
```

Try checking that the first solution really does solve the equation.

```
>
>
```

Let us turn the variables **a**, **b**, and **c** back into unknown variables. We use colons to suppress the outputs from these commands since they are not very interesting.

```
> a := 'a': b := 'b': c := 'c':
```

Notice that **eqn1** goes back to being the "general" quadratic equation.

```
> eqn1;
>
```

Here is another example that highlights the difference between using **:=** (colon equal, the assignment operator) and **=** (equal sign, which is part of an equation). Let **p** be a name for the formula **x^2-1** (**p** is an abbreviation for **p**olynomial).

```
> p := x^2-1;
```

Now what do you think the next command will do? (Think about this a bit before executing the command.)

```
> p = 2;
```

It constructed an equation. It did *not* change the value of **p**.

```
> p;
```

We can solve the equation that we constructed just above.

```
> solve( %%, x );
>
```

Here are more examples of algebraic manipulations. The **factor** command can factor polynomials and the **expand** command can multiply out polynomials. In a way, these commands are opposites of each other. This first example is very easy to do.

```
> factor( x^2+5*x-6 );
```

But this next example would be very hard to do with paper and pencil.

```
> factor( x^26+x^13+1 );
```

The next example would be easy to compute by hand.

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

But the next example would be very tedious to do by hand.

```
> expand( (x+1)^8 );
```
Here is another hard example.
```
> factor( x^6-1 );
```
The next command should undo the calculation from the last command.
```
> expand( % );
```
Maple can do algebra with expressions containing more than one variable.
```
> x := 'x': y := 'y':   # Make sure x,y don't have values.
> expand( (x+y)^3 );
> factor( % );
>
```

Here is a complicated expression that we can ask Maple to simplify.
```
> r := (t^2-t)/(t^3-t)-(t^2-1)/(t^2+t);
> simplify( r );
>
```

Maple can also do symbolic manipulations with formulas involving trig, exponential and logarithm functions. For these kinds of formulas, we mostly use the **simplify**, **expand** and **combine** commands. However, with these kinds of formulas it is not always obvious which command we might want to use.
```
> cos(x)^2 - sin(x)^2;
> simplify( % );
> combine( %% );
```
Notice how the last two commands gave two different ways of "simplifying" $\cos(x)^2 - \sin(x)^2$. We can even get a third way of rewriting this expression by using the next command.
```
> factor( %%% );
```
It is not always clear in advance how one of these commands might work on a given trig expression.
```
>
```

Here is another example. The following **simplify** command does not do any simplification.
```
> simplify( sin(x+y) );
```
But the **expand** command makes a significant change in the expression.
```
> expand( sin(x+y) );
```
Similarly, the **simplify** command does not do anything with the next expression.
```
> simplify( cos(x)*cos(y) - sin(x)*sin(y) );
```
But the **combine** command does do the expected simplification.
```
> combine( cos(x)*cos(y) - sin(x)*sin(y) );
```
From the last two examples we see that for certain trigonometric expressions, **expand** and **combine** are inverses of each other and the **simplify** command does nothing at all.. Try using the **factor** command on these last several trig expressions. Does it do anything?
```
>
```

```
>
```

Here are some examples with exponential functions.

```
> combine( exp(x)*exp(y) );
> expand( exp(x+y) );
```

So once again we see that `combine` and `expand` are, in certain circumstances, inverses of each other. Notice that only the first of the following two `simplify` commands does anything (the `simplify` command does the same thing here as the `combine` command).

```
> simplify( exp(x)*exp(y) );
> simplify( exp(x+y) );
```

Another simple example.

```
> exp(x)^2;
> simplify( % );
```

Which one of `combine` or `expand` do you think will work on the expression `exp(x)^2`? Which one will work on `exp(2*x)`? What about `simplify`?

```
>
>
```

Maple can do symbolic manipulations on more than one equation at a time. Here are two equations is two unknowns.

```
> eqn1 := 2*x+4*y=6;
> eqn2 := -x+5*y=1/2;
```

We shall ask Maple to solve this system of equations, then give the solution a name and compute a decimal approximation of the solution. Notice the way the `solve` command is written here, using braces around the equations and the variables.

```
> solve( {eqn1, eqn2}, {x, y} );
> answer := %;
> evalf( % );
```

Notice that the variables `x` and `y` still do not have values. (Those are not assignment operators inside the braces.)

```
> x; y;
```

The `solve` command found the values of `x` and `y` that satisfy the equations, but the `solve` command does not give `x` and `y` those values. The `assign` command is used to assign the values from the solution to the variables.

```
> assign( answer );
> x; y;
>
```

We have seen several examples of how Maple can solve equations symbolically. After we have computed the answer symbolically, we can always use the `evalf` command to find a numeric approximation of the solution. However, there are equations that Maple cannot solve symbolically and in those cases we can only calculate numeric approximations of a solution. Let us look at an example. Here is a fairly complicated equation. We shall give it the name **eqn3**.

```
> eqn3 := -2+exp(t)=sin(t^2);
```

Let us have Maple try to solve it symbolically.

```
> solve( eqn3, t );
```

Not a very useful answer. (Look at the answer very carefully. Can you make sense out of it? Is it really "the solution" of the equation?) So let us use the **fsolve** command to tell Maple to solve this equation numerically.

```
> fsolve( eqn3, t );
```

Why is this command called "fsolve" and not "solvef"? I have no idea why. In the help pages for the **evalf** and **fsolve** commands they are described respectively as "evaluate using floating-point arithmetic" and "solve using floating-point arithmetic". So it would seem that their names should be more consistent.

Let us give our solution a name, so that we can easily refer to it in other commands.

```
> answer := %;
```

Let us check that our numeric solution really is a solution. We will plug it in to the left and right hand sides of the original equation and see if we get the same result.

```
> subs( t=answer, lhs(eqn3) );
> subs( t=answer, rhs(eqn3) );
```

We still cannot tell if these are the same number. So find their decimal values.

```
> evalf( subs( t=answer, rhs(eqn3) ) );
> evalf( subs( t=answer, lhs(eqn3) ) );
```

Close enough. (Could we have used **%** and **%%** in the last two commands? Try it.)

```
>
```

If you ask Maple to numerically solve an equation that does not have a real number solution, then Maple will return nothing. Which is kind of an odd behavior. The following equation has only complex number solutions.

```
> fsolve( x^2+1=0, x );
```

Oops! We did something wrong. A little while back in this section we gave the variable **x** a value.

```
> x;
```

Since **x** has a value the **fsolve** command did not make sense to Maple, so it generated an error message. (To Maple, the **fsolve** command looked like **fsolve(5=0, 2)** and Maple cannot solve for 2 in 5=0.) This is a common way to make a mistake in Maple, to use a variable that has a value assigned to it as if it were an unassigned variable. We fix this mistake by unassigning **x**. (We will also unassign **y**, for future use.)

```
> x := 'x';
> y := 'y';
```

Now we can give our example of an equation that does not have a real number solution; notice that Maple seems to silently do nothing.

```
> fsolve( x^2+1=0, x );
```

Of course, **solve** has a much easier time with this equation.

```
> solve( x^2+1=0, x );
```

I is Maple's notation for the imaginary number $\sqrt{-1}$.

```
>
```

To give you a sense of how tricky some symbolic calculations can be in Maple, consider the expression $a^n\, b^n$. According to the rules of algebra found in calculus books, this should simplify (or combine) to $(a\, b)^n$.

```
> combine( a^n*b^n );
> simplify( a^n*b^n );
```

Neither command worked. Let us try the other direction. The expression $(a\, b)^n$ should expand to $a^n\, b^n$.

```
> expand( (a*b)^n );
```

Again, the command did not work. But there is a way to get this last command to do what we think it should do. We shall tell Maple to make an assumption about the unknowns **a** and **b**. We will tell Maple to **assume** that they are positive real numbers.

```
> assume( a>0 );
> assume( b>0 );
```

Now **(a*b)^n** will expand to **a^n*b^n** but **a^n*b^n** still will not combine to **(a*b)^n**.

```
> expand( (a*b)^n );
> combine( a^n*b^n );
```

The tildes next to **a** and **b** are the way that Maple reminds us that we have made some kind of assumption about those variables. We can remove the assumptions from **a** and **b** by unassigning them.

```
> a := 'a'; b := 'b';
> a, b;
```

From these last examples we see how difficult it can sometimes be to get Maple to make a known algebraic transformation. Sometimes, assumptions must be made about the variables involved for the transformation to work. But this can get very tricky. We will say more about the **assume** command in a later worksheet.

```
>
```

Besides the kind of symbolic algebra that we have seen so far, Maple can also do symbolic calculus calculations. Maple can compute derivatives. First let us define a function.

```
> f := x^2*sin(3*x);
```

Derivatives are computed with the **diff** command.

```
> diff( f, x );
```

The next command computes the second derivative of **f**.

```
> diff( f, x,x );
```

We can compute any order of derivative that we want. Here is the command for computing the 7th derivative.

```
> diff( f, x$7 );
```
Here are three interesting examples using functions **g** and **h**, which have not been defined yet..
```
> diff( g(x)*h(x), x );
> diff( h(g(x)), x );
> diff( g(x)/h(x), x );
> simplify( % );
```
Do you recognize these outputs?
```
>
```

If we have a function of several variables, then Maple can compute partial derivatives.
```
> g := 1/(x^2+y^2);
```
Maple uses the same command, `diff`, for computing partial derivatives.
```
> diff( g, x );
> diff( g, y );
```
We can compute higher order partial derivatives. Here are all four of the second order partial derivatives.
```
> diff( g, x,x );
> diff( g, y,y );
> diff( g, x,y );
> diff( g, y,x );
```
Here is what a partial derivative looks like symbolically.
```
> diff( h(x,y), x,y );
```

Here is how we find antiderivatives using Maple.
```
> int( x*ln(x), x );
```
Notice that Maple does not put a +C at the end of its antiderivatives. You are expected to know that they really are supposed to be there. You can also compute definite integrals.
```
> int( x*ln(x), x=0..2 );
> evalf( % );
```
Maple has a slight variation on the `int` command. The `Int` command is called the **inert form** of `int`. The inert form of a Maple command does not do any calculation. Instead, it typesets a formula for us.
```
> Int( x*ln(x), x );
```
We can use the inert form of a Maple command to get nice looking formulas like the following.
```
> Int( x*ln(x), x ) = int( x*ln(x), x );
```
Many of Maple's symbolic calculus commands have an inert form. The inert form is always a capitalized version of the command. Try using the inert form of `diff` to get Maple to display the product rule the way it would be displayed in a calculus book.
```
>
```

Besides doing symbolic integration Maple can also do symbolic summation. Here are a couple examples.
```
> sum( i, i=1..n );
```

```
> factor( % );
> sum( i^2, i=1..n );
> Sum( i^2, i=1..n ) = factor(%);
>
```

Maple can also compute limits.

```
> limit( sin(2*x)/(3*x), x=0 );
> Limit( ln(x^2)/(x-1), x=1 ) = limit( ln(x^2)/(x-1), x=1 );
> Limit( 1/x-1/(exp(x)-1), x=0 ) = limit( 1/x-1/(exp(x)-1), x=0
  );
>
```

Maple can do one sided limits.

```
> Limit( x^(p/ln(x)), x=0, right ) = limit( x^(p/ln(x)), x=0,
  right );
```

Maple can also do limits at infinity.

```
> Limit( (ln(n))^(a/n), n=infinity) = limit( (ln(n))^(a/n),
  n=infinity);
> Limit( x^n/exp(x), x=infinity)  = limit( x^n/exp(x),
  x=infinity);
>
```

And Maple can compute Taylor series.

```
> taylor( sin(x), x=0 );
> taylor( sin(x), x=Pi/2, 10 );
> taylor( ln(x), x=1 );
> taylor( exp(x)*cos(x), x=0 );
>
```

Sometimes `taylor` does not work.

```
> taylor( sqrt(x)*exp(x)*cos(x), x=0 );
```

But the results of `series` can be quite odd.

```
> series( sqrt(x)*exp(x)*cos(x), x=0 );
>
```

Maple has quite a lot more symbolic algebra and calculus commands. We have looked at examples of just the most important ones. Here is a list of the most important Maple commands for doing symbolic algebra calculations.

```
> ?factor
> ?simplify
> ?combine
> ?expand
> ?solve
> ?assume
```

And here is a list of the most important commands for doing symbolic calculus.

```
> ?diff
> ?int
> ?sum
> ?limit
> ?taylor

>
```

1.3.3. Maple is a graphing calculator.

It is easy to graph basic functions in Maple and the graphs are more detailed than what a graphing calculator can do. Here is the basic plotting command for Maple.

```
> plot( x^2, x=-5..5 );
```

We give the `plot` command the formula that we want to graph and then tell it what interval of numbers to draw the graph over. Here are a few more examples.

```
> plot( x^2, x=1..5 );
> plot( sin(x), x=-2*Pi..2*Pi );
> plot( x^2*cos(x), x=-2..2 );
>
```

Maple can even plot graphs from negative infinity to infinity. This is useful when we want an overall idea of what a graph looks like without having to guess at an appropriate domain.

```
> g := (x^2+3*x)/(x^4+x+1);
> plot( g, x=-infinity..infinity );
```

The above graph is not the "exact" graph of the function `g`. In order to draw a graph from negative infinity to infinity, Maple needs to distort the graph of the function in some way. But the above graph does give us a very good qualitative idea of what the graph of `g` looks like and what its main features are. Here are two more graphs that give us a sense of how Maple might distort a function when it draws a graph from negative infinity to infinity.

```
> plot( x^2, x=-infinity..infinity );
> plot(sin(x), x=-infinity..infinity );
>
```

We can also control the range used on the vertical axis of a graph.

```
> plot( sin(x), x=-2*Pi..2*Pi, -1/2..1/2 );
```

Notice that the second range did not have a variable assigned to it. If we do assign a variable to the second range, then Maple uses that variable as a label for the vertical axis. Compare the next graph with the previous one. (Look carefully, the label is pretty small.)

```
> plot( sin(x), x=-2*Pi..2*Pi, z=-1/2..1/2 );
>
```

We can give a function a name first and then graph it.

```
> f := x^3-x;
> plot( f, x=-5..5 );
```

```
> plot( f, x=-5..5, -10..10 );
```

Why is the second graph better than the first?

```
>
```

We can graph more than one function at a time by putting a list of functions inside a pair of brackets.

```
> plot( [x^2, x^2+1, x^3], x=-1..1 );
> plot( [sin(x), cos(x)], x=0..2*Pi );
```

When we graph more than one function at a time, Maple chooses colors for the graphs. But we can assign specific colors to each function by putting a list of colors inside another pair of brackets, like this.

```
> plot( [sin(x-Pi/4), sin(x), sin(x+Pi/4)], x=0..2*Pi,
        color=[red, black, blue] );
```

Maple will match each function with a color in the order that they are given. Assigning colors to graphs can sometimes help in figuring out which graph is which. Match the functions with their graphs in the next example (try to make a guess first).

```
> plot( [sin(x)*sin(4*sin(x)),
         sin(x)*sin(5*sin(x)),
         sin(x)*sin(7*sin(x))], x=0..2*Pi );
```

The next command brings up a help page with a list of Maple's predefined color names.

```
> ?plot,color
>
```

Exercise: There are four functions in the following graph (which goes from zero to infinity). Which function is which? What does this graph tell you about these functions? Which aspects of this graph are misleading?

```
> plot( [x^3, x^(1/2), ln(x), 2^x], x=0..infinity );
>
```

Let us look at a common way to make a mistake when drawing graphs with Maple. Suppose we give a value to the variable `t` somewhere in our worksheet.

```
> t := 2;
```

Now suppose that sometime later in the worksheet we give Maple the following command.

```
> plot( t^3-t, t=-5..5 );
```

Notice how this generates an error message. What we have just told Maple to do is plot the function that is constantly equal to 6 (which is what `t^3-2` equals when `t` is 2) with *2 ranging from -5 to 5* (i.e. `t=-5..5`). It is this last part that quite obviously does not make sense to Maple; how can 2 range for -5 to 5? We solve this problem by unassigning `t`.

```
> t := 't';
> plot( t^3-t, t=-5..5 );
```

Now we can state the following rule of thumb for working with Maple.

If you get a strange error message in Maple, one of the first things you should *always check is that the variables you are using as unknowns are really unknown and do not have a value assigned to them.*

Maple can make graphs of data points. When we graph data points we can have Maple either graph the points and connect them together with straight lines or just graph the points by themselves. Here is how to give Maple a list of points to be plotted and joined by straight lines.

```
> plot( [ [0,1/2], [1/2,1], [1,1/4], [3/2, 2], [2, 1] ] );
```

If we want to just plot the points with no connecting lines, then we use the `style` and `symbol` options to the `plot` command. We set the `style` option to be `point` and we set the `symbol` option to be one of `box`, `cross`, `circle`, `point`, or `diamond`.

```
> plot( [ [0,1/2], [1/2,1], [1,1/4], [3/2, 2], [2, 1] ],
        style=point, symbol=diamond );
```

Try changing the `symbol` option in the last command. Also, if we set the `style` option to `line`, that is the same as first command where the data points were connected by straight lines. (Try it.)

```
>
```

Using lists of data points, we can get Maple to draw any polygon in the plane.

```
> plot( [ [0,2], [1,-1], [-1,1], [1,1], [-1,-1], [0,2] ],
  axes=none );
```

We can give the list of points a name before we plot it.

```
> List := [ [1,1], [3,2], [3,1], [1,2], [1,1] ];
> plot( List, 0..3, 0..2 );
```

Try removing the two ranges from the last `plot` command to see what happens. Also, suppose you use a variable name in one of the ranges (as in `u=0..3`). What does that do? (Look carefully.)

```
>
```

We can give a graph a title using the `title` option to the `plot` command. The title will appear at the top of the graph.

```
> plot( 4*x*(1-x), x=0..1, title="The logistic map." );
```

We can also create graphs with labels in them, but the process is a bit involved. First we draw the graph that we wish to include a label in.

```
> plot( [4*x*(1-x), x], x=0..1, color=[red, blue],
        title="The logistic map." );
```

Now use the next command to give this graph a name. We can use any name that we like. Notice the colon at the end of this command.

```
> graph1 := %:
```

Now we use the `textplot` command to draw a graph that contains only the labels. We need to give the `textplot` command the labels we want and the coordinates of where each label should be placed.

```
> plots[textplot]( [ [0.9, 0.9, "y=x"], [0.77, 0.75, "fixed
```

```
point"] ],
                    align=RIGHT );
```

Now use the next command to give this graph a name.

```
> graph2 := %:
```

Finally, we use the **display** command to combine our last two graphs together.

```
> plots[display](graph1, graph2);
```

If a label should turn out to be in the wrong place, go back and modify the **textplot** command, re-execute it, rename it, and then re-execute the **display** command. Try adding a third label to this last example or moving one of the two labels around.

```
>
```

When we used the **textplot** and **display** commands we called them **plots[textplot]** and **plots[display]**. This is because those commands are part of the **plots** package. A **package** is a group of related Maple commands. To use a command from a package, we have to either include the package name as part of the command's name or we have to "load the package". We load a package using the **with** command. So for example, the following command loads the **plots** package.

```
> with(plots);
```

Notice that when we load a package, Maple gives us a list of all the commands in the package. Look carefully and you should see **textplot** and **display** in the list of commands. Now we can refer to the **display** command without using the package name as part of **display**'s name.

```
> display(graph1, graph2);
```

The **display** command can be used to combine almost any graphs together into one graph. It is usually used to build up a complicated graph out of several simple graphs.

```
>
```

Maple can draw graphs of functions using other coordinate systems in the plane. Here is a graph of a function $r = f(\theta)$ in polar coordinates. This graph is called a cardioid.

```
> plot( 1+sin(theta), theta=0..2*Pi, coords=polar );
```

Compare the last graph with the next one that uses the same function, but now in Cartesian coordinates.

```
> plot( 1+sin(theta), theta=0..2*Pi );
```

Both of the last two commands graphed the function $f(\theta) = 1 + \sin(\theta)$, but the first command used the (r, θ)-plane in polar coordinates and the second command used the (x, y)-plane in Cartesian coordinates (but it called the horizontal axis the θ-axis instead of calling it the x-axis).

```
>
```

Here is a more interesting graph of a function in polar coordinates.

```
> sin(theta)+sin(5*theta/2)^3;
> plot( %, theta=0..4*Pi, coords=polar );
>
```

Here is a graph of a circle as the graph of a constant function in polar coordinates.

```
> plot( 1, theta=0..2*Pi, coords=polar );
```

That graph may have looked elliptical, not circular. Maple quite often does not use the same scale on the horizontal and vertical axes. The `scaling=constrained` option in the next example "constrains" Maple to use the same scale on both axes, which makes the graph look like a circle.

```
> plot( 1, theta=0..2*Pi, coords=polar, scaling=constrained );
```

What graph would we get if we removed the `coords=polar` option from the `plot` command?

```
>
```

Maple can draw "graph paper" for us in polar coordinates.

```
> plots[coordplot]( polar, scaling=constrained );
```

Maple can draw graph paper in Cartesian coordinates also.

```
> plots[coordplot](cartesian, scaling=constrained,
  color=[black,black]);
```

The `color` option takes two colors, one for each variable in the coordinate system. Try changing the colors in the polar graph paper.

```
>
```

Maple can draw graphs of parametric curves. The following `plot` command draws a circle defined as a parametric curve (in Cartesian coordinates).

```
> plot( [cos(t), sin(t), t=0..2*Pi], scaling=constrained );
```

How would you draw an ellipse (with `scaling=constrained`)?

```
>
```

Now notice the subtle difference between the last command and the next command. The next command is almost the same, but it has the range outside the brackets, not inside.

```
> plot( [cos(t), sin(t)], t=0..2*Pi, scaling=constrained );
```

When the range is inside the brackets, Maple is graphing the parametric curve

$$x(t) = \cos(t) \quad \text{and} \quad y(t) = \sin(t) \quad \text{for} \;\; 0 < t \text{ and } t < 2\pi$$

but when the range is outside the brackets, Maple is graphing two separate functions

$$f(t) = \cos(t) \;\; \text{and} \;\; g(t) = \sin(t).$$

The difference between these last two `plot` commands is *very* important.

```
>
```

Here is a more interesting parametric curve.

```
> plot( [t+2*sin(2*t), t+2*cos(5*t), t=-2*Pi..2*Pi] );
```

Try moving the range outside of the brackets. Also try changing the values of the constants.

```
>
```

Maple can also draw parametric curves in polar coordinates. Here is another way to draw a circle.

```
> plot( [1, theta, theta=0..2*Pi], coords=polar,
  scaling=constrained );
```

(What would the graph look like if we removed the option `coords=polar` from the `plot` command? Why?) Parametric curves in polar coordinates are not very common. Most calculus books do not even mention them.

```
>
```

Maple has still another way to graph a circle. Maple can draw a circle as the graph of an equation.

```
> plots[implicitplot]( x^2+y^2=1, x=-2..2, y=-2..2,
                       scaling=constrained );
```

Notice that Maple has a specific command for graphing equations. When we ask Maple to graph an equation in *x* and *y*, we are asking Maple to plot all of the points in the *xy*-plane that solve the equation. Here is another example of graphing an equation.

```
> plots[implicitplot]( x^3+y^3=5*x*y-1/4, x=-3..3, y=-3..3 );
```

Notice how the graph of one equation can be in several pieces.

```
>
```

Maple can draw three dimensional graphs of functions of two variables.

```
> plot3d( sin(x+y), x=-Pi/2..Pi/2, y=-Pi/2..Pi/2 );
```

Be sure to click on the above graph and try moving it around with the mouse button pressed down. Here are a few more examples.

```
> plot3d( cos(x*y) , x=-3..3, y=-3..3 );
> plot3d( sin(sqrt(x^2+y^2)), x=-6..6, y=-6..6 );
> plot3d( sin(sqrt(x^2+y^2)), x=-6..6, y=-6..6, style=line );
> plot3d( sin(sqrt(x^2+y^2)), x=-6..6, y=-6..6, style=contour
  );
> plot3d( sin(sqrt(x^2+y^2)), x=-6..6, y=-6..6, style=point,
  color=black);
```

Notice that the changes made in the last four examples can also be done by clicking on one of the graphs and then clicking on appropriate buttons that appear in the "context bar" at the top of the Maple window.

```
>
```

Here are some more functions of two variables.

```
> f := (x^2+y^2)*exp(-x^2-y^2);
> plot3d( f, x=-3..3, y=-3..3 );
```

Be sure to look "underneath" this last graph. The next function is very similar but the graph is quite a bit different. Why?

```
> f := (x^2+5*y^2)*exp(-x^2-y^2);
> plot3d( f, x=-3..3, y=-3..3 );
```

Here is one more example.

```
> f := 1/( 1 + sqrt(abs(y-sin(x))) + sqrt(abs(x+sin(y))) );
> plot3d( f, x=-1..1, y=-1..1 );
>
```

Maple can graph functions of two variables using other coordinate systems. Here is a sphere as the graph of a constant function $\rho = f(\theta, \phi)$ in spherical coordinates

```
> plot3d( 1, theta=0..2*Pi, phi=0..Pi, coords=spherical,
          scaling=constrained );
```

Maple can also graph parametric surfaces in space. Here is a sphere graphed as a parametric surface (in rectangular coordinates).

```
> plot3d( [cos(u)*sin(v), sin(u)*sin(v), cos(v)], u=0..2*Pi,
  v=0..Pi,
          scaling=constrained);
```

Here is the same sphere as a parametric surface using spherical coordinates in space.

```
> plot3d( [1, theta, phi], theta=0..2*Pi, phi=0..Pi,
          coords=spherical, scaling=constrained);
```

We should mention that the sphere can also be graphed as the solution set of an equation in three unknowns.

```
> plots[implicitplot3d]( x^2+y^2+z^2=1, x=-1..1, y=-1..1,
  z=-1..1,
                         scaling=constrained);
>
```

Here is a more interesting graph of a function of two variables in spherical coordinates. This is a "bumpy sphere" that you can look inside of (by rotating the graph).

```
> rho := 1 + .1*cos(5*theta)*cos(5*phi);
> plot3d( rho, theta=0..2*Pi, phi=0..3*Pi/4, coords=spherical
  );
```

How would you close the hole at the bottom of the sphere?

```
>
```

Here is an interesting parametric surface in space (using rectangular coordinates).

```
> plot3d( [(2+sin(v))*cos(u), (2+sin(v))*sin(u), u+cos(v)],
          u=0..4*Pi, v=0..2*Pi );
```

How would you get more spirals to show in the graph?

```
>
```

One of the amazing things that Maple can do is create animations using the **animate** command. Here are three very simple examples of animation. To view an animation, execute the Maple command and after the graph becomes visible click on it. Some VCR type buttons will appear at the top of the Maple window in the "context bar". Click on the "play" button. Try playing with the other buttons also.

```
> plots[animate]( sin(x-p), x=0..2*Pi, p=0..2*Pi );
> plots[animate]( sin(p*x), x=0..2*Pi, p=1..4 );
> plots[animate]( {p+sin(x), -p+sin(x), (p^2)*sin(x)},
  x=0..2*Pi,
                 p=(sqrt(5)-1)/2..(sqrt(5)+1)/2, color=gold );
>
```

Maple creates animations by computing and storing 16 separate graphs of the given function (each graph is called a **frame**) and then displaying the 16 graphs very quickly in the order they were computed. For each frame in the animation the parameter `p` is incremented a little bit, and this is what makes each frame a little different from the frame just before it (the parameter `p` can be given any name you like). There is a button in the "context bar" for controlling how many "frames per second" are drawn during the animation, the range is from as slow as one frame per second up to as fast as 30 or 40 frames per second. There is also another button that lets you step through the frames one by one, so you can see how they change. The `animate` command has a `frames` option for specifying how many frames Maple should compute for the animation (if you want more than the default 16 frames). By specifying a fairly high number for the `frames` option you can make either very smooth or very long animations, depending on how long the range for the frame parameter is and how many frames per second are displayed. The next animation has a lot of frames, so it is fairly smooth (and the frame parameter is `mu`).

```
> f := x -> mu*x*(1-x);
> plots[animate]( f(f(f(x))), x=0..1, mu=1.. 4,
                 numpoints=150, frames=100, color=blue );
>
```

The next animation also has a lot of frames and it runs for a fairly long time. It is also periodic. There is a button for "continuous play" and if you click on that button, because of the periodicity this animation it will play smoothly over and over again.

```
> plots[animate]( [(1+.5*sin(t)*cos(5*s))*cos(s),
                  (1+.5*sin(t)*cos(5*s))*sin(s), s=0..2*Pi],
                 t=0..2*Pi, scaling=constrained,
  numpoints=100,
                 color=blue, axes=none, frames=100 );
```

This last animation is not as complicated as it seems. It is "morphing" a circle. The parametric equations in the `animate` command define a "circle" whose radius (given by the term `1+sin(t)*.5*cos(5*s)`) changes both with angle (the `s` variable) and with time (the `t` variable which is also the frame parameter). Try changing any of the parameters in the command. Even if you do not yet understand the equations, you can modify them to see what happens. Small changes in the equations can lead to big changes in the animation.

```
>
```

The next animation is based on the previous one. Some of the constants have been changed and two circles are being morphed simultaneously.

```
> plots[animate]( {[(1+2*sin(t)*cos(6*s))*cos(s),
                    (1+2*sin(t)*cos(6*s))*sin(s), s=0..2*Pi],
                   [(1+2*sin(t)*cos(6*s))*cos(s),
                    (1-2*sin(t)*cos(6*s))*sin(s), s=0..2*Pi]},

                   t=0..2*Pi, scaling=constrained,
  numpoints=150,
                   color=blue, axes=none, frames=100 );
>
```

The rest of this section is made up of more examples, without much explanation, of drawing graphs with Maple. First there are some two dimensional curves, both parametric curves and functions in polar coordinates. Then there are several three dimensional graphs.

Here are two different kinds of spirals as parametric curves. (Try converting these to graphs of functions in polar coordinates.)

```
> plot( [t*cos(t), t*sin(t), t=0..2*Pi], scaling=constrained );
> plot( [exp(t)*cos(5*t), exp(t)*sin(5*t), t=0..2*Pi],
        scaling=constrained);
```

Longer versions of the two previous spirals.

```
> plot( [t*cos(t), t*sin(t), t=0..10*Pi], scaling=constrained
  );
> plot( [exp(t)*cos(10*t), exp(t)*sin(10*t), t=-2*Pi..10*Pi],
        scaling=constrained );
```

The next spiral is called Cornu's Spiral (also called Euler's Spiral or clothoid).

```
> plot( [FresnelS(x), FresnelC(x), x=-4..4],
        scaling=constrained, axes=none, title=`Cornu's Spiral`
  );
>
```

The next example is a Lissajous curve. Try different values for **a**, **b**, and **c** (but see the comment next to **d**).

```
> a:=3/4:  b:=0:  d:=4:  # d is the denominator of a.
> plot( [sin(a*t+b*Pi), sin(t), t=0..2*Pi*d],
  scaling=constrained );
>
```

The next two parametric curves are surprising for their elegance. They have interesting names too. First, an epitrochoid.

```
> a:=3:  b:=1:  c:=2:  d:=.15:
> d*((a+b)*cos(t)-c*cos((a+b)*t/b)),
  d*((a+b)*sin(t)-c*sin((a+b)*t/b));
```

```
> plot( [ %, t=0..2*Pi ], scaling=constrained, axes=none,
  title=`Epitrochoid` );
```

Next, a hypotrochoid.

```
> a:=4:  b:=1:  c:=2:  d:=.2:
> d*((a-b)*cos(t)+c*cos((a-b)*t/b)),
  d*((a-b)*sin(t)-c*sin((a-b)*t/b));
> plot( [ %, t=0..2*Pi ], scaling=constrained, axes=none,
  title=`Hypotrociod` );
>
```

The next curve is from "Mathematics Magazine", Vol. 69, No. 3, June 1996, pages 185-189. Try changing some of the constants in the parametric equations.

```
> plot( [cos(t)+1/2*cos(7*t)+1/3*sin(17*t),
>        sin(t)+1/2*sin(7*t)+1/3*cos(17*t),
>       t=0..2*Pi], scaling=constrained );
>
```

Here are several curves that are graphs of functions in polar coordinates.

A cochleoid.

```
> sin(theta)/theta;
> plot( %, theta=-6*Pi..6*Pi, coords=polar,
  scaling=constrained, title=`Cochleoid` );
```

A butterfly curve.

```
> exp(sin(theta))-2*cos(4*theta);
> plot( %, theta=0..2*Pi, coords=polar, scaling=constrained,
  title=`Butterfly curve` );
```

Another butterfly curve.

```
> exp(sin(theta))-2*cos(4*theta)+(sin(theta/12))^5;
> plot( %, theta=0..20*Pi, coords=polar, scaling=constrained,
  numpoints=1000, title=`Butterfly curve`);
```

A few unnamed curves.

```
> sin(theta)+sin(5*theta/2)^3;
> plot( %, theta=0..4*Pi, coords=polar, scaling=constrained,
  axes=none );

> sin(6*theta/5);
> plot( %, theta=0..10*Pi, coords=polar, scaling=constrained,
  axes=none );

> theta*cos(theta);
> plot( %, theta=-19*Pi/2..19*Pi/2, coords=polar,
  scaling=constrained, axes=none );
>
```

Now for some more 3D plots.

The next example is a parametric surface using rectangular coordinates.

```
> [r*cos(phi), r*sin(phi), phi];
> plot3d( %, r=0..1, phi=0..6*Pi, grid=[15,45],
  orientation=[55,70], style=patch, shading=zhue, axes=frame );
```

Here is another parametric surface using rectangular coordinates.

```
> [ (1-u)*(3+cos(v))*cos(4*Pi*u),
    (1-u)*(3+cos(v))*sin(4*Pi*u),
    3*u+(1-u)*sin(v) ];
> plot3d( %, u=0..1, v=0..2*Pi, orientation=[-14,76] );
>
```

The next command uses spherical coordinates in space. The radius `rho` is a function of the angles `theta` and `phi`.

```
> rho := (1.3)^theta*sin(phi);
> plot3d( rho, theta=-1..2*Pi, phi=0..Pi, coords=spherical);
```

If you do not remember what spherical coordinates are, Maple can help you remember by drawing a surface with constant coordinate value for each of the three coordinates.

```
> plots[coordplot3d]( spherical );
>
```

The next example uses cylindrical coordinates in space. The radius `r` is a function of angle `theta` and height `z` (but since `r` does not depend explicitly on `theta`, this graph has rotational symmetry).

```
> r := (5*cos(z)^2-1)/3;
> plot3d( r, theta=0..2*Pi, z=-Pi..Pi, coords=cylindrical,
  style=patch, color=r );
```

Here is a demonstration of cylindrical coordinates.

```
> plots[coordplot3d]( cylindrical );
```

The next example is a parametric surface using cylindrical coordinates in space.

```
> [z*theta, theta, cos(z^2)];
> plot3d( %, theta=0..Pi, z=-2..2, coords=cylindrical,
  color=theta );
>
```

The next two examples use a specialized Maple command called `tubeplot`.

```
> plots[tubeplot]( [ -10*cos(t) - 2*cos(5*t) + 15*sin(2*t),
                     -15*cos(2*t) + 10*sin(t) - 2*sin(5*t),
                      10*cos(3*t) ],
                   t= 0..2*Pi, radius=3*cos(t*Pi/3) );
```

You can change any of the following five parameters to get different graphs.

```
> a:=2: b:=4/5: c:=1: m:=4: n:=7:
> r := a + b*cos(n*t):
> z := c*sin(n*t):
> curve := [ r*cos(m*t), r*sin(m*t), z ];
> plots[tubeplot]( curve, t=0..2*Pi, radius=1/4, tubepoints=20,
  numpoints=200, orientation=[45,10], style=patch,
  shading=xyz);
>
```

Not all of Maple's graphs come from equations.

```
> plots[polyhedraplot]( [0,0,0], polytype=dodecahedron,
  orientation=[71,66], style=patch, scaling=constrained );
```

The next example comes from a worksheet about duality that is part of Maple's online documentation. It is a 3-D graph, so try rotating it.

```
> with(geom3d):
> GreatRhombiicosidodecahedron( t9, point(o,0,0,0), 1. ):
> duality( dt9, t9, sphere(m9, [o, MidRadius(t9)]) ):
> draw( [t9(color=maroon), dt9(color=gold)], cutout=7/8,
        lightmodel=light4, orientation=[0,32],
        title=`Dual of great rhombiicosidodecahedron` );
>

>
```

1.3.4. Maple is a programmable calculator. (Optional)

If you have some experience with programming languages, or with programming hand held calculators, then this section will give you an introduction to programming in Maple. If you do not have any experience with computer programming, then you can skip this section for now. Several of the later worksheets will introduce you to the basics of programming.

Maple has most of the common programming language features. Maple has an assignment statement, a very versatile repetition statement, a conditional statement, procedures, and many built in data types.

We have already seen many examples of Maple's assignment statement which uses the assignment operator (`:=`).

The basic form of Maple's looping command is

for *index_variable* **from** *initial_value* **to** *final_value* **by** *step_size* **do**
sequence_of_Maple_commands
end do

Here are some simple examples.

```
> for i from 0 to 8 do (1+x)^i end do;
```

```
> for i from 0 to 8 by 2 do (1+x)^i end do;
>
```

The next two commands will differentiate `x*cos(x)` for us ten times.

```
> f := x*cos(x);
> for n from 1 to 10 do f := diff(f, x) end do;
```

Notice how in the last example, while we can see all ten derivatives of `x*cos(x)`, we cannot refer to them because they were all give the same name `f`. Here is a way to fix this. Maple can generate sequential names on the fly by using something called the **concatenation operator** (which looks like this, `||`).

```
> f0 := x*cos(x);
> for n from 1 to 10 do f||n := diff(f||(n-1), x) end do;
```

Notice from the output that the concatenation operator `||` is not part of the variable names. This operator separates the letter part of the variable name, which is fixed, from the numeric part of the variable name, which Maple computes on the fly. Now `f7` is the name for the seventh derivative of `x*cos(x)`.

```
> f7;
>
```

There are many other forms of Maple's loop statement. Here is a form that steps its way through all the elements of some data structure.

for *index_variable* **in** *data_structure* **do** *sequence_of_Maple_commands* **end do**

The following command counts how many letters are in each word from a list.

```
> for fruit in [plum, apple, pineapple, grape, watermelon] do
    length(fruit)
  end do;
```

The next command differentiates the terms of a polynomial one at a time.

```
> for term in 1 + 3*x + 12*x^2 + x^3 + 7*x^4 do
    diff(term, x)
  end do;
>
```

Maple also has a while loop.

while *boolean_expression* **do** *sequence_of_Maple_commands* **end do**

Here is the general form of Maple's conditional statement.

if *boolean_expression* **then**
  sequence_of_Maple_commands

```
elif boolean expression then
    sequence_of_Maple_commands
else
    sequence_of_Maple_commands
end if
```

There can be as many `elif/then` clauses as needed. However, the `elif/then` part and the `else` part of the conditional statement are optional. So we also have the following two simpler forms of the conditional statement.

```
if boolean_expression then
    sequence_of_Maple_commands
else
    sequence_of_Maple_commands
end if
```

and

```
if boolean_expression then
   sequence_of_Maple_statements
end if
```

Here is an example of a conditional statement that computes a piecewise defined function.

```
> x := 3/2;
> if   x <= 0 then x^2
  elif x <= 1 then x^3
  elif x <= 2 then x^4
  else x^5
  end if;
>
```

The following combination of a loop and an if-statement will print out all of the prime numbers less than 100. Notice how indentation can be used, much like in any other programming language, to make the code easier to read.

```
> for i from 1 while i <= 100 do
    if isprime(i) then
      print(i)
    end if;
  end do;
```

Try to modify the above example so that it also counts how many primes there are less than 100. What if you wanted to know how many primes there are less than 1,000,000?

Maple allows us to use **procedures** to group statements together and to pass them parameters. Here is a simple procedure that finds the larger of two numbers.

```
> bigger := proc(x, y)
```

```
      if x > y then x
      else y
      end if
   end;
```

Here **bigger** is the name we give to the procedure. The definition of the procedure begins with **proc** and ends with **end**. The formal parameters to the procedure are placed inside parentheses right after the **proc**. This procedure has two formal parameters. This procedure has only one statement in its definition (a conditional statement), but there can be any number of statements. The return value of a procedure is whatever is the last expression computed by the procedure.

Now that we have defined the procedure we can call it. The actual parameters in the procedure call are placed inside parentheses right after the name of the procedure.

```
> bigger( 10, 11 );
>
```

Procedures can also have local variables. The next procedure finds the average of a list of numbers.

```
> avg := proc( L )
      local N, sum, x;     # declare three local variables
      N := nops( L );      # count the number of operands in list
   L
      sum := 0;            # initialize sum
      for x in L do
         sum := sum + x;   # compute a running total
      end do;
      sum/N;               # compute and return the average
   end;
```

Notice the use of indentation and comments to make the procedure understandable.

```
> L := [1, 1, 3, 4];
> avg( L );
>
```

Almost all Maple commands are actually Maple procedures written in the Maple programming language. Take for example the **isprime** command. Let us try and look at the definition of this procedure.

```
> print( isprime );
```

This did not tell us much. Here is a way to get Maple to be more forthcoming.

```
> interface(verboseproc=2);
> print( isprime );
> interface(verboseproc=1);
```

Obviously, this definition is quite complex. But this shows us two things. Maple commands are programs written in the Maple language, and we can examine, even modify, these commands if we know enough about Maple.

```
>
```

The last major piece of Maple's programming puzzle is Maple's built in data types and data structures. But this is too big a subject to summarize here. There is a whole worksheet later on just about data structures and data types.

The following command will bring up a help page listing all the keywords used in Maple's programming statements.

```
> ?statement
```

Basic information about writing procedures can be found in the following two help pages. (Unfortunately, these two help pages are not easy reading.)

```
> ?procedure
> ?parameter
```

And links to more information about Maple programming can be found in the next help page.

```
> ?index,procedure
>

>
```

1.4. Getting help in Maple

Maple has several built in ways for getting help about both Maple commands and the Maple graphical user interface.

To begin with, there is Balloon Help. With Balloon Help you can point (but not click) the mouse at any of the buttons or menu items at the top of the Maple window and get at least a little bit of information about it. This helps you get started with Maple's user interface. By default, Balloon Help should be turned on, but if is not, you can turn it on by going to the "File => Preferences.." menu item, clicking on the "General" tab at the top of the Preferences window, and then clicking on the "Balloon Help" check box near the bottom of the Preference window.

You should work your way through the "New User's Tour". To start it, click on the "New User's Tour" item in Maple's "Help" menu.

The "Introduction" menu item at the top of the "Help" menu is a good place to start reading about Maple if you want to get more in depth about how it works. In particular, notice the **Help Browser** that appears at the top of the "Introduction to Maple 8" help page (just below the tool bars). The Help Browser is a very well structured outline of the whole Maple help documentation. As you click on an item in any column, that item is either expanded into the next column or displayed in the window below the Help Browser. The Help Browser appears at the top of every help page.

If you want information about a specific Maple command, at a Maple prompt type a question mark followed by the command. This will bring up a help window that you can read and then close in the usual way.

```
> ?plot
```

Notice that this command did *not* need a semi colon at the end of it.

A very quick way to get help on a specific Maple command is to put the cursor on the command in question and then press the F1 key (found at the top of the keyboard). This will bring up the command's help page right away. Go ahead and try this on the (incorrect) **plot** command below.

```
> plot(x^2, -1..1);
```

Notice how the Help Browser at the top of the help page shows where you are in the outline of the help documentation. The Help Browser can be very useful for finding information closely related to whatever help page you are currently reading.

At the end of the help page for every command there are examples of using the command. It is often a good idea to use "cut and paste" to cut the examples out of the help page and paste them into your worksheet, where you can try them out and make changes to them. For example, call up the help page on **polyhedraplot** using the next command, cut the example from the end of the help page and paste it into this page at the next empty prompt, and then execute the example. (Be sure to click on the outputs and try rotating them.)

```
> ?polyhedraplot
>
>
```

At the very end of many help pages there is a section called "See Also". This is a list of **hyperlinks** to related help pages. Clicking on a hyperlink takes you to the associated help page. For more information about Maple's hyperlinks, follow this hyperlink or execute the following command.

```
> ?worksheet,documenting,hyperlinks
```

If you are trying to find out how Maple handles some mathematical concept and you do not have any idea what Maple command might be useful, then you need to **search** the Help System. You can do this three ways. First, you can try to browse the Help System using the Help Browser. This is not very efficient, but it can sometimes be very informative. Otherwise you need to use either the "Topic Search..." or the "Full Text Search..." menu items in the "Help" menu. It is hard to say which of these searches is better to use. Try the Topic Search first; if what you are looking for is in it, this will be the fastest way to find it. But Topic Searches often fail, and then you have to use the Full Text Search. There is more information about searching in the "Using Help" menu item in the "Help" menu.

```
>
```

Below are a few commands that call up help pages that you might find useful. The first command will bring up an introduction to Maple. (This is the same as using the "Introduction" item at the top of Maple's "Help" menu.)

```
> ?introduction
```

The next command will bring up Maple's help for the Help System. (This is the same as using the

"Using Help" item in Maple's "Help" menu.)

```
> ?worksheet,reference,HelpGuide
```

The use of **?** to call up help pages is described, not surprisingly, in the next help page.

```
> ?
```

The next command brings up a Maple worksheet that is part of the New User's Tour. This worksheet is another brief introduction to Maple's online help.

```
> ?newuser,topic11
```

The next command will bring up help on using Maple's worksheet interface.

```
> ?worksheet
```

In particular, look at the sub-sections "Getting to Know the Worksheet Interface" and "Documenting Your Work". And also read the next section of this worksheet.

The next command will bring up a glossary of Maple terms (this is the same as the "Glossary" item in the "Help" menu).

```
> ?glossary
```

The next command will bring up an index to all of Maple's built in library functions.

```
> ?index,functions
```

The next command brings up a list of Maple's mathematical functions.

```
> ?inifcn
```

The next command brings up a list of all the Maple packages.

```
> ?index,package
```

The next command will bring up an index to all of Maple's commands.

```
> ?index
>

>
```

1.5. Working with worksheets

The following subsections go over the details of some particularly important aspects of working with worksheets.

```
>
```

1.5.1. Entering text into a worksheet

Maple worksheets are divided into three kinds of areas, input areas, output areas, and text areas. Input is displayed in red type, output in blue type, and text in black type. When you open a new (empty) Maple worksheet it begins with a prompt which marks an input area. After you type in a command and execute it, Maple displays the results of the command in an output area and then it gives you a new prompt for another input. By default, Maple will then alternate between input and output regions. If you want to create a text area to put some comments in, then you have to do something different.

There are two ways to put comments like these in a Maple worksheet. First, you can convert a Maple input prompt from "Maple mode" to "text mode". To do this, you click on an empty Maple prompt and then use the "Insert => Text" menu item to enter "text mode" (or you can click on the big T button in the middle of the tool bar, or type the key combination Ctrl-T). Once in "text mode" the prompt disappears and you can begin to type in your text. Notice that in "text mode" what you type is in black, and when you are in "Maple mode" what you type is in red. Put the cursor at the prompt below, enter "text mode", and then type in some text. You can type in as many lines of text as you like. Notice that the lines of text stay inside of a bracket on the left hand edge.

>

If you need a new Maple command prompt somewhere in a worksheet (for example, right after a comment that you put in, as in the last paragraph), place the cursor at the end of the line just above where you want the Maple command prompt and use the "Insert => Execution Group =>After Cursor" menu item (or use the "[>" button on the tool bar, or type Ctrl-J). Try putting a command prompt after the text that you entered above and also in the empty space just below this line.

The other way to enter some text into a worksheet is to put the cursor about where you want the text area and then use the "Insert => Paragraph => After" menu item (or type Ctrl-Shift-J). This creates a text area just below where you placed the cursor.

There is a slight difference between the two ways of entering text. The first way puts the text inside an "execution group" (see below). This means that there is a thin vertical bracket on the left side of the text area. The second method does not place the text inside an execution group, so there is no vertical bracket around the left side of the text area. Most of the text in these worksheets was entered using the second method.

>

1.5.2. Menu bar, tool bar, and context bar

If you look at the top of the Maple window, you see three rows of menus and buttons just below the title bar. The first (i.e. top) row is called the **menu bar**, the second row is called the **tool bar**, and the third row is called the **context bar**.

The menu bar is where you access all of Maple's menu items; these are mostly commands for manipulating Maple's graphical user interface. None of Maple's mathematical commands are in the menus. If you look through the menus, you will notice that many of the menu items have "keyboard shortcuts" associated to them. It is worth while to learn a few of the shortcut keys for commonly used menu items; they save a lot of time and trouble from always using the mouse. For example, the cut, copy, paste and undo items in the "Edit" menu have the standard Ctrl+x, Ctrl+c, Ctrl+v and Ctrl-z keyboard shortcuts respectively. Another important shortcut key is Ctrl+s for the menu item "File => Save". Get in the habit of using this keyboard shortcut to save you worksheets as you are working. Maple has a tendency to crash, so you want to save your work often (at least every 10 minutes) to minimize the chance of losing your time and work. (Here is a list of many other keyboard shortcuts.)

The menu bar changes depending on where the cursor is positioned in a worksheet. The following help page lists each of the different kinds of menu bars and provides information about every item in each menu.

```
> ?worksheet,reference,menus
```

The tool bar provides one-click shortcuts to a few of the most important menu items. Use Balloon Help to find out what each tool bar button does, or look at the following help page.

```
> ?worksheet,reference,toolbar
```

The tool bar for help pages is different than the tool bar for worksheets. In particular, the help tool bar has several navigational buttons (buttons with arrows on them). The following page describes the tool bar for help pages.

```
> ?worksheet,reference,helptoolbar
```

The context bar is called the context bar because this row changes depending on the context of where the cursor is placed. For example, if the cursor is placed on this line, then the context bar displays items for working with text. If the cursor is placed on a Maple command line (try the one just below), then the context bar changes. If you place the cursor on the output from the command, then the context bar changes again.

```
> factor(x^2-4);
```

$$(x-2)(x+2)$$

There are quite a few different context bars in Maple; we cannot go into the details of every one. Whenever you want to know what something on a context bar does, first use Balloon Help to get a rough idea and then do a "Topic Search" (look at the menu item "Help => Topic Search...") to get more details. The following help page lists each of the different kinds of context bars and provides a bit of information about every button on each context bar.

```
> ?worksheet,reference,contextbar
```

After the text area and command line context bars, the other important context bars are for plots. There are four different graphics context bars, one each for 2-dimensional plots, 3-dimensional plots, 2-dimensional animations, and 3-dimensional animations. Execute each command below, and after executing it, click on the output plot and look at the context bar. Try all of the buttons in each of these context bars.

```
> plot(sin(x), x=-Pi..Pi);
> plot3d(sin(x)*cos(y), x=-Pi..Pi, y=-Pi..Pi);
> plots[animate](sin(t*x), x=-Pi..Pi, t=1..4, frames=50);
> plots[animate3d](sin(t*x)*cos(t*y), x=-Pi..Pi, y=-Pi..Pi,
  t=1..2, frames=50);
```

Notice one obscure little detail. Each two and three dimensional animation actually has two context bars, one as a two or three dimensional plot and another as an animation. When you first click on an animation, you get the animation context bar. But if you look carefully at the far right hand edge of the context bar you will see two small up and down arrows. Clicking on these two arrows lets you switch between the two context bars. Try this with one of the above two animations.

>

Finally, notice that in the "View" menu in the menu bar, the first two items let you turn on or off the displaying of the tool bar and the context bar. A check mark next to an item means that that bar is displayed.

>

1.5.3. Maple Notation vs. Standard Math Notation

Notice how in the next Maple command the appearance of the input command and the appearance of the output result are very different.

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

$$\sqrt{2}$$

The input is in a form that Maple refers to as **Maple Notation** and the output is in a form referred to as **Standard Math Notation** (Standard Math Notation is also referred to as Typeset Notation). In Maple's usual mode of operation, you enter commands at the prompt in Maple Notation and the results are printed out in Standard Math Notation. You can change this if you want (using the "I/O Display" tab from the "File => Preferences..." menu item). You can, if you wish, enter commands in Standard Math Notation and also have Maple print the results in Maple Notation. But the usual mode of operation is certainly the most convenient and I would not recommend changing it.

>

There is one situation in which you might want to enter something in Standard Math Notation. If you want to put a formula in some text and you want the formula to look nice, like this $x^2\, y + \sqrt{x}\, \cos(\theta)$ instead of looking kind of ugly, like this x^2*y+sqrt(x)cos(theta), then you need to enter the formula using Standard Math Notation. The basic idea for using Standard Math Notation in the middle of some text is that you type Ctrl-r to switch Standard Math Notation (or click on the "sigma button" in the tool bar, or use the "Insert => Standard Math" menu item), then start typing Maple Notation (what you type will appear in a box that appears in the context bar), then type Ctrl-t to return to "text mode" (or click on the "T button" in the tool bar, or use the "Insert => Text" menu item), and at this point Maple will translate (or "render") the Maple Notation that you just typed into Standard Math Notation that will appear inside your text. With a bit of practice this is pretty easy to do.Try using your mouse to highlight the formula in Standard Math Notation just below and then look up at the context bar to see the Maple Notation. Try editing the Maple Notation in the context bar, then type Ctrl-t to see your changes appear in the Standard Math Notation below (the changes do not appear until you type Ctrl-t).

$$\int (3\, x^2 - 2\, x + 27)\, \mathbf{e}^{\alpha} \cos(x - \alpha)\, dx$$

You can also try to highlight just a small piece of the above formula (try the exponential) and change just that piece. Now try to enter your own formula in Standard Math Notation into the blank space below.

The following command brings up a very brief help page about using Standard Math Notation.

```
> ?worksheet,documenting,insstdmath
>

>
```

1.5.4. Execution groups

If you want to combine several consecutive Maple command prompts together to form an execution group, put the cursor on the first of the command prompts and then hit the F4 key. Each time you hit F4, another command prompt will join the execution group. If you want to split up an execution group, put the cursor on the prompt that you want to break away from the group and hit the F3 key. Try combining and breaking up the following sequence of commands.

```
> 100!;
> ifactor(%);
> expand(%);
>
```

You can also join or split execution groups by using the "Edit => Split or Join => Join Execution Group" and the "Edit => Split or Join => Split Execution Group" menu items.

Recall that you can create new Maple prompts by using the key combination Ctrl-j. Try creating several empty prompts after this line and then combining them into a single execution group.

For some more information about execution groups, read the help page called up by the following command.

```
> ?worksheet,documenting,executiongroups

>
```

1.5.5. An important warning

Here is an important fact about using Maple worksheets. *Maple does not remember anything from one session of Maple to another*. What does this mean? Look at the next Maple command. It defines `five` to have the value 5, and the result of that command is shown below the command. **Do not execute the next command.** Instead, skip to the command just below it and execute that command.

```
> five := 5;
```

$$five := 5$$

```
> five;
```

When you execute the second command, you see that Maple does not know that `five` represents 5. Even though the output of the first command is sitting there, it does not mean that Maple remembers that result from the last Maple session. In order for this Maple session to know that **five** represents 5, you have to actually execute the Maple command.

Here is even more confusing example. Look at the following commands and their "outputs", but **do not execute them**.

```
> one := 1;
```

$$one := 2$$

```
> two := 2;
```

$$two := 1$$

Originally, the commands were `one:=2;` and `two:=1;`. After I executed them I went back and "fixed" the commands to be `one:=1;` and `two:=2;`, but I did not re-execute the commands after I changed them. So the outputs are left over from the original commands and do not reflect the commands that are there now. And if you execute the following command, you see that in this session Maple does not yet know any value for the variables `one` and `two`.

```
> one, two;
```

Here is one more example. Look at these commands and their outputs, but **do not execute them**.

```
> a := 3;
```

$$a := 3$$

```
> solve( a*x+5=0, x );
```

$$\frac{-5}{17}$$

```
> a := 17;
```

$$a := 17$$

The output of the second command does not seem to be correct; the solution of $3\,x + 5 = 0$ is not -5/17. But then we see in the third command that the value of `a` becomes 17, and the solution of $17\,x + 5 = 0$ is -5/17. What happened is that I executed the first of these three commands, then the second one, then the third one, and then I went back and re-executed the second one (out of laziness say, instead of reentering the command below the third command). Then I saved the Maple worksheet in a mixed up state that can be very confusing. This kind of mixed up state of the worksheet is quite common with Maple; it is up to you work with your worksheets in an organized way to try and prevent this kind of confusion.

```
>
```

The moral of these examples is that you have to be very careful when you interpret the outputs in a Maple worksheet. You must make sure that all of the Maple commands are executed in the current Maple session and that they are executed in the correct order. To help you with this, near the bottom of Maple's Edit menu you will find an "Execute =>" menu item containing two sub items, "Selection" and "Worksheet". The latter can be used to execute all the commands in a worksheet in their proper order form beginning to end. The former can be used to update just a selected section of a worksheet.

```
>
```

```
>
```

1.5.6. The `restart` command

Sometimes you might get confused and you cannot remember which variables you have assigned values to, or even Maple might get confused and start giving you strange error messages or strange results. If this should happen, use the **`restart`** command to make Maple act as if you had just started it. When you execute the **`restart`** command, Maple does not do anything to the appearance of your worksheet, but Maple will forget the values that you have assigned to any variables. After a restart you can reuse variable names without worrying about which ones already have values. In addition, when you use the **`restart`** command Maple will "forget" some of its internal information that might be causing Maple to give you strange results. Using the **`restart`** command is almost, but not quite, the same as saving your worksheet, quitting Maple, and then restarting Maple with your worksheet (which, in certain extreme cases, may be exactly what you need to do to get Maple to recover from a case of strange behavior).

If you execute the **`restart`** command below, you will see that it seems to have no outward affect on your worksheet.

```
> restart;
```

A very quick and convenient way to restart Maple is to click on the extreme right hand button on the tool bar (the button has an icon of an arrow looping back to its tail). You can use balloon help to verify that this button restarts Maple.

The following command will bring up the help page for the **`restart`** command.

```
> ?restart
>
```

```
>
```

1.5.7. Working without a worksheet

Working with previously written worksheets is not the only way to use Maple. If you start up Maple without a worksheet (for example, by just clicking on the Maple item in the Windows "Start Menu"), then Maple begins with a "blank" worksheet, a worksheet with a single command prompt on it waiting for a command to be entered. This mode of using Maple is very close to how you would use a calculator; you just type in commands and get results, one after the other. The worksheet that you are creating when you do this is sometimes called a "scratch sheet". It is kind of like doing paper and pencil work on a scratch pad. If you look at the title bar at the top of the Maple window, you will see the name "untitled" for the worksheet. Maple always puts the name of the open worksheet in the title bar, but a scratch worksheet does not have a title. When you are done working with Maple in this way you can either discard the contents of the worksheet or save your scratch worksheet as a permanent file for later reference (but be careful about the "important warning" given in Section 1.5.5 above). If you choose to save the worksheet (by typing Ctrl-S or using the "File -> Save As..." menu item), Maple will open the "Save As..." dialog box in which you give the worksheet a name and tell Maple where to save it (be sure that you pay attention to where you are saving the worksheet).

>

1.5.8. Working with multiple worksheets

You can open more that one worksheet at a time with Maple. For example, you may have a previously written worksheet like this one open and want to open a blank scratch pad worksheet to try out an idea. Or you may want to compare the contents of two previously written worksheets. To do either of these you use the "File" menu at the top of the Maple window.

>

To open a blank worksheet, use the "File => New" menu item (or type Ctrl+N). To open a previously written worksheet, use the "File => Open..." menu item (or type Ctrl+O). Once you have two (or more) worksheets open, you can switch between them using the "Window" menu. Every currently open worksheet is listed at the bottom of the "Window" menu. The worksheet that is active (i.e., displayed in front of all the other worksheets) has a check mark next to its name in the list. Go ahead and try opening a new worksheet and then switching between this worksheet and the new one.

Another way to switch between open worksheets (or between open worksheets and open help pages) is to type Ctrl+Tab. Each time you type Ctrl+Tab Maple will jump to the next open worksheet. If you have a lot of worksheets open and you want to step through all of them, just hold down the Ctrl key and strike the Tab key repeatedly to cycle through the worksheets.

>

It is important to know that Maple has two different modes of working with multiple worksheets. In one mode, everything that you type in one of the open worksheets is also remembered in all of the other open worksheets. In the other mode, the open worksheets are all independent of each other. The first mode is called "single server mode" and the later mode is called "parallel server mode". What "single server mode" means is that, for example, if you define the variable **x** to be 5 in one open worksheet, then **x** is 5 in all the other open worksheets also. This can get confusing, and is not always what you want. Unfortunately, single server mode is the default mode for Maple. To use parallel server mode you have to use the "Parallel Server Maple 8" icon in the "Maple 8" program group from the Windows "Start Menu". However, if you start up Maple by double clicking on a worksheet (and this is the most common way to start Maple), then you are in single server mode and I do not know an easy way to change this.

How can you tell if you are in single server mode or parallel server mode? Look at the title bar of the Maple window for each open worksheet; the phrase [Server n] appears after the name of each worksheet (where n is the number of the server). If every worksheet has the same server number n, then you are in single server mode. If the worksheets have different server numbers, then you are in parallel server mode. The following help command brings up a brief explanation of kernel modes.

```
> ?worksheet,reference,kernelmodes
```

```
>
```

1.5.9. Save your work!

Maple has a tendency to crash. When it does, you will lose all of the work you have put into your worksheet since the last time you saved it. So get in the habit of saving your worksheet as you work on it. Do not wait until you are done to save your worksheet! You save a copy of your current worksheet using the "File => Save" menu item or, better yet, the Ctrl+S keyboard shortcut. This shortcut key makes it almost effortless to save your work as you are typing along. Save your work at least every 10 minutes. One of the easiest places to save a copy is on the Windows desktop. That way it is easy to find when you are done.

```
>
```

```
>
```

1.5.10. More information

The next command brings up a Maple worksheet that is part of the New User's Tour. This worksheet is a brief introduction to Maple's worksheet environment.

```
> ?newuser,topic02
```

The following command will bring up a help page on using Maple's worksheet interface.

```
> ?worksheet
```

In particular, look at the sub-section The Worksheet Interface.

```
>
```

```
>
```

1.6. How Maple is organized

Maple is a large, complex system and Maple has a tremendous number of commands and functions for doing mathematics. No one can know everything about all of the Maple commands. As you learn more about mathematics you will want to learn more about Maple so that you can put it and your mathematical knowledge to good use. Knowing something about the overall structure of Maple and the organization of all its commands will help you when you want to learn about using Maple in some new way.

```
>
```

Every Maple command is of one of two types. A small number of Maple commands are **built in commands** and all of the rest of the commands are **library commands**. The library commands are further classified as either **main library commands** or**package commands**.

The built in commands are part of what is called the Maple kernel. The kernel is the actual computer program that runs Maple. There are just a few Maple commands that are built in. An example of a built in command is **diff**, the command that differentiates expressions. If we ask Maple to print the definition of one of the built in commands, Maple does not tell us much about the command other than that it is built in.

```
> print( diff );
```

When you start Maple on your computer, the Maple kernel and all of the built in commands are loaded into your computer's main memory. The Maple kernel and the built in commands form the foundation for all of the rest of the Maple system. In particular, the Maple kernel defines the Maple programming language.

```
>
```

The great majority of Maple's commands are **library commands**. Library commands differ from built in commands in two ways. First of all, the library commands are written in the Maple programming language while the built in commands are written in the C programming language. We will see shortly what this difference means and why it is important.

The other difference between library commands and built in commands is that unlike the built in commands, the library commands are not loaded into memory automatically when Maple starts up. The reason for this is that if Maple loaded all of its commands automatically when you started the Maple program, your computer probably would not have enough memory to run Maple. And in truth, very little would be gained by loading everything at once, since you will never use more than a small fraction of Maple's commands in any given Maple session. By not loading most commands until they are needed, no sacrifice is made in terms of Maple's versatility, but a huge gain is made in terms of efficiency.

As we mentioned earlier, the library commands are divided into two subgroups, the main library commands and the package commands. What distinguishes these two subgroups is how the commands in them are loaded into Maple.

The main library commands are commands that are automatically loaded into Maple whenever you use them. So main library commands are just as convenient to use as built in commands. The main library commands are fairly numerous. These are the commands that most Maple users use most of the time.

An example of a main library command is **ln**, the logarithm function. If we ask Maple to print out the definition of this command, it does not tell us much (but at least we see that **ln** is not a builtin command).

```
> print( ln );
```

If we make use of an additional Maple command, then Maple will give us the whole definition of the **ln** command.

```
> interface( verboseproc=2 );
> print( ln );
```

What you are looking at is a program written in the Maple programming language. This Maple code is what lets the **ln** command do symbolic manipulations like the following one. (Can you find the line of Maple code that actually is responsible for this manipulation?)

```
> ln( 1/2 );
```

Now we can explain the most important difference between built in commands and library commands. Every library command is really the name for a Maple program, just as **ln** is the name for the above program (though most Maple commands have much longer programs than **ln**). Every one of these programs can be examined using the **print** command. Once you know enough about the Maple language, you can read these programs and find out how the great majority of the Maple system works. In addition, you can modify these programs to extend Maple (or fix bugs in Maple). This ability to extend Maple is one of its most important characteristics. Researchers in specialized fields can write or modify Maple commands to solve highly specialized problems. The only limitation to Maple's adaptability is the fact that the Maple kernel and the built in commands are written in the C programming language and their code is neither modifiable nor accessible.

Later in these worksheets we will examine the code for several Maple commands and we will see just how they are able to do many of their symbolic manipulations. We will also look at a few examples of extending Maple's abilities in some simple ways.

```
>
```

The bulk of Maple's library commands are package commands. A **package** is a collection of Maple commands that are somehow related. For example, the **linalg** package is a collection of commands for working with linear algebra, the **LinearAlgebra** package is another, newer, collection of linear algebra commands, the **numtheory** package is a collection of commands for working with number theory, and the **DEtools** package is a collection of commands for working with differential equations.

To use a package command you have to either explicitly load the command or you have to load the whole package that the command is in. Here is an example. The **student** package is a collection of commands that are useful to students taking calculus. One of the commands in this package is **leftbox**, which draws a picture of the left hand sums that approximate a definite integral. Suppose we try to use this command.

```
> leftbox(sqrt(x), x=0..3, 10);
```

Maple just repeated back to us what we entered. This is because the name **leftbox** is still undefined to Maple so Maple does not know how to evaluate it. We need to load the **leftbox** command into Maple. There are two ways to do this. First, we can use the **long name** for **leftbox**, which is **student[leftbox]**, in our command.

```
> student[leftbox](sqrt(x), x=0..3, 10);
```

The long name of a package command loads the command just for that one use. If you are going to use a package command several times, or you are going to use several commands from the same package, you should load all of the commands in a package into Maple's memory using the **with** command.

```
> with(student);
```

Notice that the `with` command returns the names of all the commands in the package that was loaded. Now that the `student` package has been loaded, we can use the short name of `leftbox`.

```
> leftbox(sqrt(x), x=0..3, 10);
>
```

Some packages are further divided into subpackages. For example, the `Student[Calculus1]` package is another (newer) collection of commands for students taking calculus and it is a **subpackage** of the `Student` package. One of the commands in `Student[Calculus1]` is `RiemannSum`, which can draw pictures of many different kinds of Riemann sums. Suppose we try to use this command.

```
> RiemannSum( sqrt(x), x=0..3, method=left, partition=10,
  output=plot );
```

Maple just repeated back to us what we entered. The name `RiemannSum` is still undefined to Maple; we need to load the `RiemannSum` command into Maple. We can either use the (rather long) long name for `RiemannSum`, which is `Student[Calculus1][RiemannSum]`, in our command.

```
> Student[Calculus1][RiemannSum]( sqrt(x), x=0..3, method=left,
  partition=10, output=plot );
```

Or we can load all of the commands in the `Student[Calculus1]` package into Maple's memory using the `with` command.

```
> with(Student[Calculus1]);
```

Now that the `Student[Calculus1]` package has been loaded, we can use the short name of `RiemannSum`.

```
> RiemannSum( sqrt(x), x=0..3, method=left, partition=10,
  output=plot );
>
```

Our first use of the `leftbox` and `RiemannSum` commands above demonstrates an important thing to remember when using Maple. If you enter a Maple command and Maple just repeats it back to you, without any error message, this sometimes means that the command is undefined to Maple. This can be caused by one of two things. Either you misspelled the name of the command or you are using a package command that has not been loaded yet. The easiest way to tell which one of these cases you are in is to click on the command and then hit the F1 key at the top of the keyboard. This will cause the Maple help system to search for the command. If the search comes up empty, you have a misspelled command. If the search is successful, then look at the top of the help page. Every Maple help page begins with the long name of the command, so this will tell you which package the command is in.

```
>
```

The following help page will bring up an index of all the packages in the Maple library with hyperlinks to more information about each package.

```
> ?index,packages
```

The following help pages give an overview of the (new) `Student[Calculus1]` package and the (old) `student` package and they have hyperlinks to all of the commands in the packages.

```
> ?Student[Calculus1]
> ?student

>
```