3 MAPLE LANGUAGE

Sections

Objectives

After reading this chapter, you should be able to:

- Distinguish elements of the Maple language
- Use Maple operators in accordance with operator precedence and associativity
- Choose Maple names according to variable-naming conventions
- Classify the kinds of Maple statements
- Decompose expressions into expression trees
- Manage variable assignments
- Unassign previously assigned names
- Describe Maple's rules for automatic simplification and full evaluation

3.1 Introduction

Understanding how Maple interprets commands will greatly assist your work. This section introduces key features of the Maple language that you will use to solve a variety of problems throughout this text and, hopefully, in your studies.

3.1.1 Language

Maple commands constitute a written ***language***. A written language provides a means to communicate concepts using combinations of symbols in a defined and "universally" accepted fashion. For instance, the English language combines words and punctuation to form sentences. Grammar dictates the rules for constructing sentences. From properly constructed sentences, readers may then hopefully extract meaning, assuming the writer was not incompetent or crazy.

3.1.2 Maple Language

When speaking of Maple, consider Maple's words as commands, functions, and other symbols formed from standard keyboard characters. As with other languages, other symbols punctuate Maple's "sentences." To write useful commands in Maple, the user must still consider grammar and meaning, often referred to as ***syntax*** and ***semantics***, respectively:

- Syntax governs the structure of commands and command entry
- Semantics govern how the program understands and acts upon commands

For example, refreshing a worksheet involves a one-word "sentence" composed of a known command (**restart**) punctuated by a semicolon (**;**):

Step 25: Restart Maple

```
> restart;
```

Refresh the current worksheet by erasing all current assignments.

As discussed in this chapter, you build expressions using elements of Maple's language.

3.2 Language Elements

Using a set of characters to write Maple commands, you use the following items which comprise the written-language elements of Maple: *tokens*, *token separators*, and *escape characters*.

3.2.1 Characters

Maple's "alphabet" uses standard keyboard characters:

- Uppercase letters: **A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**
- Lowercase letters: **a b c d e f g h i j k l m n o p r s t u v w x y z**
- Digits: **0 1 2 3 4 5 6 7 8 9**
- Miscellaneous characters: **@ # $ % ~ _ | ^ * + - = & , . ? ! : ; " ' ‘ \ / ( ) [ ] { } < >** *space*

If you are unfamiliar with some of the character names, you should review Appendix A. This "alphabet" provides the characters used to build the language elements, discussed below.

3.2.2 Tokens

Maple's ***tokens*** form many of the language's "words," which help you build commands, as you would construct a sentence out of words. Table 3-1 summarizes the kinds of tokens Maple uses. From these elements, you may construct all kinds of important expressions, as demonstrated throughout this text. This chapter provides an overview of operators, names, reserved words, and some punctuation. Chapter 4 delves further into integers, punctuation, and strings by showing you how to build expressions with these elements.

Table 3-1 Maple Tokens

Token	Definition	Reference
Operator	symbol that performs mathematical tasks	`?operator`
Name	label for functions, mathematical variables, and system variables	`?name`
Reserved Word	reserved and unassignable command names	`?keyword`
Integer	whole numbers that use the digits `0`, `1`, `2`, `3`, `4`, `5`, `6` , `7`, `8`, and `9`	`?integer`
Punctuation	symbol that separates tokens: `; : , () {} []`	`?;` or `?:`, `?,`, `?list`
String	characters enclosed by double quotes (`""`)	`?string`

3.2.3 Token Separators

Besides punctuation, Maple distinguishes language elements with token separators that include spaces and new lines:

- Enter spaces by pressing the spacebar on your keyboard.
- Enter new lines by pressing Enter or Return (↵).

Use as many token separators as you wish, but never break apart an element, like the assignment operator (`:=`)!

3.2.4 Escape Characters

Escape characters shown in Table 3-2 resemble tokens but serve a different purpose. An escape character tells Maple to perform a task “away from” the command line. Maple “leaves” the command line, activates or initiates the required task, and then “returns.” For instance, Maple evaluates commands after the prompt until encountering a `#`, which tells Maple to treat all proceeding characters as inert commentary.

Table 3-2 Escape Characters

Character	Some Uses	Example	Reference
`?`	invoke help, find help on topic	`?sqrt`	`?help, ??`
`#`	skip comments	`# hello`	`?comment`
`\`	continue line, insert control character	`'a+\nb'`	`?backslash, ?\`
`!`	execute operating system command	`!ls`	`?escape`

Practice!

1. Identify the following pieces of input in terms of Maple-language elements: `1`, `+`, `TEST`, `sin`, `and`, `:`, `()`, `"Test Plot"`.
2. Does the input `A := 1` produce an error message?
3. Why does the input `A : = 1` produce an error message?

3.3 Operators

This section focuses on the ***operator*** token. An operator is a mathematical symbol that performs tasks on zero or more expressions, as shown in this section.

3.3.1 Arithmetic

Maple uses the common notation for arithmetic operations shown in Table 3-3. For instance, the expression 1 + 2 combines the integers 1 and 2 with the addition operator, the plus sign (+).

Elements acted upon by operators are called ***operands***. For instance, try an expression that uses addition and division:

Step 26: Arithmetic

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

Add fractions together. Remember to press ↵ from now on!

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

Maple added 1 *to* $\frac{1}{2}$, *which produces* $\frac{3}{2}$.

Consult **?arithop** for more information on arithmetic operations.

Table 3-3 Arithmetic Operators

Operation	Symbol	Standard Math	Maple Notation
Exponentiation	**^** and ******	$2^3 = 8$	**2^3** **2**3**
Multiplication	*****	$2 \times 3 = 6$	**2*3**
Division	**/**	$2 \div 3 = \frac{2}{3}$	**2/3**
Addition	**+**	$2 + 3 = 5$	**2+3**
Subtraction	**-**	$2 - 3 = -1$	**2-3**

3.3.2 Read Me!

Suppose you wish to multiply **a** by **b**. Never forget to specify operators! Never enter **ab**. Never enter **a(b)** or **(b)a** or **(a)(b)**. Never even *think* of entering **a times b**. However, *do* enter **a*b** or **b*a**. As discussed below, you can also specify parentheses, as in **(a)*b**, **a*(b)**, **(a)*(b)**, and **(a*b)**.

3.3.3 Functional Syntax

Occasionally you might see an alternative syntax that Maple allows for operators using a functional form. For instance, rather than entering **1+2**, you may enter **'+'(1,2)**. In general,

the functional syntax is **`'operator'(operands)`**. Why should you bother? This syntax helps you form expression trees, which are discussed in Section 3.7.2.

3.3.4 Miscellaneous Operators

Consult **`?index[expression]`** or **`?operator`** for a complete operator list. Consult **`?operators[binary]`**, **`?operators[unary]`**, and **`?operators[nullary]`** for listing according to operator type. Consult **`?define`** to create custom operators.

Practice!

4. In Maple Notation, determine the operators that the expression $-1+2^3$ employs.
5. Assign to *a* the value 1. Assign to *b* the value 2. Multiply *a* and *b*. Now, enter *ab*. Compare the output for entering **`a*b`** and **`ab`**.
6. Enter $4 \le 5$ and $5 \ge 4$ into Maple. Compare the output from both inputs. Hint: Consult **`?inequality`**.

3.3.1 Operator Precedence

Beware of ***operator precedence***. Operator precedence dictates that certain operators always act upon expressions *before* other operators do. For example, multiplication and division precede addition and subtraction. Perhaps you don't believe me? Suppose you incorrectly enter the expression $\frac{1}{2+3}$ such that you ignore the rules of operator precedence:

Step 27: Treat Operators with Care!

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

Attempt to solve the problem $\frac{1}{2+3}$.

$$\frac{7}{2}$$

Why does Maple not produce $\frac{1}{5}$?

The expression **1/2+3** first evaluates **1/2** and then adds **3** to the result! Instead, you must surround **2+3** with parentheses to ensure Maple's evaluation of $\frac{1}{5}$:

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

Use parentheses to solve the problem $\frac{1}{2+3}$.

$$\frac{1}{5}$$

Now you have the correct answer!

You may always surround ambiguous expressions with parentheses, but never use square brackets (**[]**) , curly braces (**{}**), or angle brackets (**<>**) when you need parentheses! Consult **?operators[precedence]** and **?syntax** for more information.

3.3.2 Operator Associativity

Operator associativity resolves the treatment of operators that have equivalent precedence. Many operations are *left associative*, like subtraction (**-**), which means that they compute *left* to *right*. For instance, the expression $1-2-3$ evaluates to -4 because $1-2$ will be calculated before reaching -3. To demonstrate, enter the input without parentheses:

Step 28: Operator Associativity

```
> 1-2-3;
```

Subtract three numbers. Which order does Maple pick for the operations?

$$-4$$

Maple automatically evaluates $(1-2)-3=-4$ by default.

Next, change the order of Maple's operations by surrounding **2-3** with parentheses:

```
> 1-(2-3);
```

Subtract three numbers. Deliberately supply parentheses.

$$2$$

Maple now evaluates $1-(2-3)=2$.

Some operators, like exponentiation (**^**), require parentheses if the operators are used in conjunction with another operator. For instance, you must enter **2^(-2)** to express 2^{-2}. These kinds of operators are called *non-associative*.

Practice!

7. What Maple input will produce the following output:

$$\frac{a^2}{A+\frac{1}{a}}?$$

8. What happens if you enter **sin((a+b);** ? What correction should you make?

9. Will **[1+3]/4** produce the same output as **(1+3)/4**? Why or why not?

10. Is **1^2^3** acceptable Maple input? Hint: Consult **?operators[precedence]**.

3.4 Names

Equations, like $y = mx + b$, use letters as variables that represent "changeable" quantities. Maple expressions are often constructed from such variables. This section reviews variable naming conventions using the name token.

3.4.1 Symbols

Maple's language defines the most basic form of a variable as a ***symbol***, which follows these rules:

- You may construct symbols using lowercase letters (**a**-**z**), uppercase letters (**A**-**Z**), digits (**0**-**9**), and the underscore character (_).
- You never use a digit as the first character of symbol.
- Maple is case sensitive. Consult **?type[symbol]** for more information.

3.4.2 Name

When speaking symbolic mathematics, a Maple ***name*** typical serves the role of a variable. A name is a symbol with the additional property of *indexing*, as discussed in Chapter 4. Maple already uses many names to label functions and constants. You will use names to build and assign expressions. Examples of valid names include **A**, **A1**, **A11**, **A1A**, **A_1**, **a**, and **alpha**. As a brief example, enter an unassigned name, which Maple evaluates as just the name you entered:

Step 29: Names

```
> Name;                                   Check the value of name Name.
                                  Name     Name has no assigned value, so Maple reports the name.
```

Consult **?name** or **?symbol**, **?type[name]**, and **?indexed**. for a complete set of rules and more information.

3.4.3 Protected names

Maple does not allow you to use every possible name. Imagine the potential chaos if you decided **sqrt** should not perform a square root! To prevent you from making potentially disastrous assignments, Maple predefines certain names, called ***protected names***, for library functions,

variables, and constants. Maple prevents protected names from assignments. You have already seen examples of other protected names, such as **sin** and **sqrt**. For a list of what you cannot assign, investigate **?ininames** and **?inifcn**. You should also avoid trying to reassign keywords, as discussed in Section 3.5.

Rather than skimming multitudes of Help windows, a user typically, and inadvertently, discovers protection during a session:

Step 30: Check for Protected Names

```
> D := 1;                                                    Attempt to assign D.
Error, attempting to assign to `D` which is protected        Maple reports an error.
```

Otherwise, you may check for protection with **type(*name*,protected)**:

```
> type(D,protected);                                         Check if Maple protects D from assignment.
                                  true                       Maple confirms that D is protected.
```

Also, try entering **?*name*** to see if Maple already uses ***name*** for a function or command name.

```
> ?D                                                         Maple opens a Help window on D.
```

If you really, *really* want to use a protected name for an assignment, investigate **?protect** or **?unprotect** (but don't tell anyone I told you).

3.4.4 Backquotes

To create more descriptive names, surround characters with backquotes, as in **‘A‘**. You will usually find the backquote (‘ or **‘**) on the same keyboard key with a tilde (~). Backquotes help you create names that Maple would normally consider illegal, such as **`1A`**, and **`Even I am a name!`**. Such names are also available for assignments:

Step 31: Assign Backquote Names

```
> ‘Variable 1‘ := 10;                                        Assign a backquoted name to another.
```

$$Variable\ 1 := 10$$ Maple reports your assignment.

In general,

- Do not use forwards quotes (’, `’`, or `'`), which are found on the same key with a double quote (“, ”, or `"`).
- You may backquote some reserved words, but should avoid doing so.
- For an unprotected name or unreserved word, Maple automatically replaces that name with the actual name written without backquotes.

For further rules, consult `?name`.

3.4.5 Miscellaneous

After you have become familiar with Maple, you will likely encounter many items related to Maple names. These items include local, global, and environment variables; programming variables; indexed names; aliases; assumptions; labels; variables with assumptions; and spreadsheet variables. You will discover these items as you work through this book.

Practice!

11. Which of the following is a valid Maple name, `1` and/or `‘1‘`? Hint: Enter `type(expr,name)`.
12. What input will generate *Success* := *Practice* + *Patience* as output?
13. Are the Maple names Ira and ira equivalent? Why or why not?
14. Is γ protected? Hint: Refer to the symbol palette.

3.5 Reserved Words

Maple does not consider reserved words, also called *keywords*, as protected names. Unlike a protected name, Maple never allows you to change a reserved word's meaning because the word forms an important element of the Maple language. For instance, some reserved words include operators, like **and** and **union**. Many reserved words are used for programming, as demonstrated in Appendix E. Consult **?keyword** for a listing of Maple's reserved words. If you attempt to assign a reserved word, you will receive an error message:

Step 32: Reserved Words

```
> and := 10;                                   Attempt to assign reserved word and.
Error, reserved word `and` unexpected          Maple reports an error.
```

As discussed in **?name**, you may backquote a reserved word to give a semblance of an assignable name, but I do not recommend doing so.

Practice!

15. Are **in** and **or** reserved words?

16. Can you assign values to **in** and **or**?

3.6 Statements

Recall the language analogy for a moment. In a written language, a portion of text called a *sentence* usually ends with a punctuation character, as in ending an English sentence with a period. Well, Maple's sentences are called ***statements***. A statement combines tokens to form a meaningful portion of input terminated by a semicolon (**;**) or colon (**:**). Continuing the analogy, you might even wish to think of a worksheet as a "book," with a collection of "sentences."

Maple processes each statement in succession, where each statement instructs Maple to perform a task. Overall, Maple classifies eight categories of statements described briefly in Table 3-4. For a majority of this text, you will concentrate on expression and assignment statements. In later studies, you will need the selection and repetition statements which help you develop programs. Investigate `?index[statement]` for a listing help topics and commands that deal with statements.

Table 3-4 Maple Statements

Statement	Definition	Example	Reference
expression	Entering an expression instructs Maple to evaluate the expression.	`> sqrt(4);`	`?entering`
assignment	Maple will evaluate ***expr*** and store the results in ***name***.	`> x := sqrt(4);`	`?:=`
empty	Maple will do nothing and skip to the next statement.	`> ;`	`?;`, `?empty`
quit	GUI: You will exit from a worksheet. Command-line: You will exit from Maple.	`> quit;`	`?quit`, Figure 1-4
selection	Choose a statement based on a condition.	`> x := 1: if x <= 1` `> then x:=2; end if;`	`?if`, `?try`, Appendix E
repetition	Repeat a statement.	`> for i from 1 to 3 do` `> i; end do;`	`?do`, Appendix E
save	Save variable assignments into a file.	`> save "work.m"`	`?save`, Appendix C
read	Read in a file into Maple.	`> read "work.m"`	`?read`, Appendix C

Practice!

17. Which portion of the input `1+1` is an expression? statement?

18. Why is an assignment not an expression?

19. Why does Maple allow multiple input statements in one paragraph?

20. How does Maple classify `restart`?

3.7 Expressions

This section elaborates further on the structure and construction of expressions. Later sections rely on your understanding of how Maple interprets expressions, so study this section carefully.

3.7.1 Expression Elements

Recall that you build an ***expression*** using Maple-language elements called tokens, which are built from Maple's character set. Generally, Maple expressions are built from names, operators, integers, and some punctuation. For instance, consider the mathematical expression, $\sqrt{1+\frac{1}{10}}$.

The Maple expression `sqrt(1+1/10)` uses the function name `sqrt` along with operators (`+` and `/`) that connect the integers `1` and `10`. Parentheses `()` punctuate this expression. Note that including a terminator as `sqrt(1+1/10);` creates an expression *statement*, a portion of input you enter that Maple will evaluate. For even more interesting expression "building-blocks" that Maple provides, check out Chapter 4. Consult `?syntax` and `?index[expression]` for more information.

3.7.2 Expression Trees

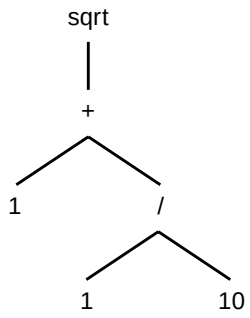

Figure 3-1 Expression Tree for $\sqrt{1+\frac{1}{10}}$

Figure 3-1 demonstrates how to draw an expression, like $\sqrt{1+1/10}$, as an ***expression tree***, as shown in Figure 3-1. (In computer science, trees tend to appear upside-down.) Following syntax rules, like operator precedence and associativity, Maple *parses*, or "splits," the expression into subexpressions. For this example, some subexpressions include $1+1/10$ and $1/10$. Maple continues parsing the subexpressions until reaching tokens. Names, operators, and integers form the nodes on the expression tree.† The branches indicate which elements are acted upon, or connected to, a particular node. Advanced students might wish to investigate `?packages[types]` for related information.

3.7.3 Expression Palette

If you need help entering expressions, refer to Sections 2.5.1-2.5.3 for information.

Practice!

21. Draw an expression tree for the expression $\frac{1}{a+b\sqrt{2.1}}$.

22. Which of the following Maple inputs `x=10`, `x`, `10`, and `sqrt(10+x/(10-x))` is an expression? Hint: Consult `?equation` for `x=10`.

23. Is the input `x:=10` an expression?

†. More formally, Maple *types* form the nodes of an expression tree. The notion of expression type is explored in Chapter 4.

24. What does the input `sqrt('+'(1,'/'(1,10)))` generate? Hint: See Section 3.3.3.

3.8 Assignments

You have already seen expression statements which help you perform calculations in an interactive session. However, what if you wish to reuse previous input or split long expressions in a series of smaller inputs? Rather than having you retype, cut, and paste expressions all the time, Maple conveniently provides for you assignment statements to clarify and ease your work. This section expands the overview from Chapter 2 and demonstrates the rules and tips for assigning expressions. Students interested in using assignments for programming should review Appendix E.

3.8.1 Assignment Syntax

Maple's general assignment statement ***name*:=*expression*** evaluates ***expression*** and stores the result inside ***name***. After entering an assignment, Maple replaces each instance of ***name*** with the results of ***expression***. For example, store the value 1 in the name A:

Step 33: Assignment

```
> A := 1;
```

Assign to the name A the value 1.

$$A := 1$$

Maple evaluates your assignment.

Note that you should never enter assignments in reverse order, e.g., `1:=A`! When used for assignment, a Maple name is sometimes called a ***variable*** because the name represents a quantity that can change in value. Maple provides different kinds of variables, as discussed in Appendix E. Consult `?:=` and `?assign` for more rules of syntax.

Professional Success: The Language of Assignments

How should you read ***name:=expr***? Sometimes Maple references will say, "assign ***name*** to ***expr***," or, "assign to ***name*** the value of ***expr***." (Refer to **?:=**.) I tend to choose the latter, but many languages actually say, "assign ***expr*** to ***name***," which might seem backwards. Why backwards? When you make an assignment, your software calculates ***expr*** first and then stores the result in ***name***. Some books will even write $name \leftarrow expr$ to indicate an assignment. With Maple, you type **:=** because there is no arrow character ←, and you may actually talk about an assignment in any of the preceding ways. But, to avoid confusion, I suggest you say "***name*** gets ***expr***" or "store ***expr*** in ***name***."

3.8.2 Assigning Expressions

The expression that you assign does not have to be numerical. You may assign any expression, including other names:

Step 34: Assigning other names

```
> Name1 := Name2;
```

Assign to the name Name1 the "value" Name2.

$$Name1 := Name2$$

Maple evaluates your assignment.

Eventually, $Name2$ might obtain a value, which will change the value of $Name1$, as discussed in Section 3.11. For now, imagine the possibilities of assigning more complicated and exotic expressions, as in **y:=m*x+b**, **z:=x^2/sqrt(y)**, and even **p:=plot(exp(x),x=1..2)**!

3.8.3 Using Assignments

After assigning a name, Maple replaces all future occurrences of the name with the assigned value. Therefore, you may enter ***name*** as an expression statement to check the value assigned to ***name***:

Step 35: Check Assignment

```
> Var := 1;
```

$$Var := 1$$

Assign to the name Var the value 1.

Maple evaluates your assignment.

```
> Var;
```

$$1$$

Check the value assigned to Var.

Maple evaluates Var which produces the value 1.

By entering **Var** as an expression statement, Maple evaluated the expression represented by **Var**. You will discover more about how Maple performs that evaluation in Section 3.11. Also, until you reassign or remove the value of Var, Maple uses *only* the assigned value 1 in place of Var:

Step 36: Maple Remembers Assignments!

```
> Var+2;
```

$$3$$

Form an expression using Var.

Maple replaced Var with the value 1.

How long will Maple remember your assignment? Until you enter **restart** or quit Maple, as discussed in Section 3.10. You should also consult **?assigned**, **?anames**, and **?unames** for checking assignments.

3.8.4 Assign and Equal are Different!

Expressions built with the equals sign (**=**) are called *equations*, not assignments! Equations are a specific type of expression, as discussed in Chapter 4. So, what happens if you enter equals (**=**) instead of the assign operator (**:=**)?

Step 37: Equations are not Assignments!

```
> test = 1;
```

$$test = 1$$

The input test=1 is an expression, not an assignment.

Maple evaluates the expression.

It may look like Maple assigned $test$, but it didn't! Check the value of $test$ as proof:

```
> test;
```

$$test$$

Check the value of test.

Maple did NOT assign test.

If you wanted to assign *test*, you should have entered **test:=1** instead. Check **?equation** to learn more about the proper use of **=**.

3.8.5 Unassigning Variables

Once assigned, a name remains assigned until the name gains another value, or the name is cleared. ***Unassigning*** removes any previously assigned expression stored in a name. Unassign ***name*** by surrounding ***name*** with forward quotes, as in ***name := 'name'***:

Step 38: Unassign a name

```
> Var:=1;
```

$$Var := 1$$

Assign to the name Var the value 1. Maple evaluates the assignment.

```
> Var:='Var';
```

$$Var := Var$$

Unassign the name Var. Unassigning Var removed the stored expression.

```
> Var;
```

$$Var$$

Is Var still assigned to any value? No: Unassigning Var cleared its value.

You might wonder why entering **Var:=Var** will not unassign **Var**. When you review Section 3.11.2, you will discover why the forward quotes work. You should also consult **?unassign** and **?evaln** for alternative approaches to unassigning names.

Practice!

25. Explain the difference between the input statements **J=72** and **J:=72**.

26. All in one statement, evaluate the expression $1 + \sqrt{4b}$ and assign the result to the name *dis*.

27. Store the expression $a + b$ in c. Assign to b the value 1. Then, store the value of $c + 1$ in d. What expression does d now store? Why?

28. Check the values of *a*, *b*, *c*, and *d*.

29. Unassign *a*, *b*, *c*, and *d*.

3.9 Maple Quotes

By now, you have seen a variety of Maple quotes which, I admit, can be quite confusing. Table 3-5 summarizes the quotes, how they may appear in Maple windows, and their uses. Refer also to `?quotes` for online help.

Table 3-5 Maple Quotes

Quote	Symbols	Use	Example	Reference
back	`‘`, ‘	form a symbol surround commands	`‘Hi Jenn!‘` `‘+‘(1,2)`	`?‘`
forward	`'`, `’`, ’	unassign delay evaluation	`x := ’x’` `’’x’’`	`?’`
double	`"`, “, ”	form a string	`"hello"`	`?"`

3.10 Worksheet Management

Assigned names may conflict with other assignments on different worksheets during the same Maple session. This section presents methods for efficient worksheet management.

3.10.1 Restart

Entering `restart` resets a Maple worksheet and removes all assignments and accessed library packages. Appendix D and later chapters demonstrate the library packages.

Step 39: Restarting a worksheet

```
> Var:=1;
```

$$Var := 1$$

Assign to the name Var the value 1. Maple evaluates the assignment.

```
> restart;
```

Restart the worksheet. All assignments are erased.

```
> Var;
```

$$Var$$

Check the value of 'Var. All assignments were erased.

For more information, consult **?restart**.

3.10.2 Kernel Modes

During a session, opening a new worksheet does not necessarily start a new session with no variable assignments! Two kernel modes determine how worksheets share name assignments:

- ***Shared-Kernel Mode***: Worksheets share *all* assignments, regardless of where the statements are entered. Entering **restart** removes all assignments from all open worksheets.
- ***Parallel-Kernel Mode***: Worksheets operate independently and do not share assignments from other worksheets. Entering **restart** removes all assignments *only* in the worksheet in which **restart** was entered.

Usually, Maple starts in shared-kernel mode. Consult **?configuring** for more details on choosing another mode.

3.10.3 Dittos

To avoid retyping many expressions, use ***dittos***, which are *nullary* operators: **%**, **%%**, **%%%**. A single ditto reissues the previously entered expression:

Step 40: Dittos

```
> a+b;
```

Evaluate the expression a+b.

$$a + b$$

```
> %;
```

Reissue the previously evaluated expression.

$$a + b$$

Maple evaluates a+b again.

I suggest avoiding dittos until you have gained more experience managing your assignments. Consult **?ditto** and **?operators[nullary]** for more information.

3.10.4 Assigned Names

To tell Maple to report all variables have been assigned during the current session, enter **anames()**. For example, in a fresh worksheet, assign some variables and then ask Maple to tell you the currently assigned variables:

Step 41: Assigned Names

```
> restart;
```

Refresh your worksheet.

```
> a:=1: b:=2:
```

Assign variables a and b.

```
> anames();
```

Ask Maple to tell you the assigned names.

$$a, b$$

Maple reports the assigned names.

For information and options, consult **?anames**. You will find related information with **?unames** and **?assigned**.

Practice!

30. Determine your system's default kernel mode.

31. Run Maple in parallel-kernel mode. Open two worksheets. Enter the statement **A:=1** in one worksheet. Check the value of A in both worksheets.

32. What output would you expect from the following Maple input statements?

```
> A:=3: B:=2: C:=1: %%%;
```

3.11 Evaluating Expressions

To fully use Maple's power you need to learn how Maple interprets your input. Why? So that you may understand how the evaluation of that input occurs. Up until now, you have seen Maple evaluating basic input using "straightforward" commands. But, what if you saw input that looked like the following?

```
> a:=1: x:=a+y: y:=1: a:=2: x;
```

If you check Maple, you will discover that the correct output is 2. If you didn't anticipate that result, don't worry – I haven't taught you how Maple evaluates expressions yet.

In general, Maple maintains exact, symbolic expressions whenever possible. Towards this goal, Maple uses built-in rules to compute output: ***automatic simplification*** and ***full evaluation***, as discussed in this section.

3.11.1 Automatic Simplification

Usually, someone performing calculations wants the simplest, or "smallest" possible results. For instance, 1/2 is simpler than 2/4. Simplification attempts to reduce expressions to smaller forms and might require ingenuity. Maple thinks like a human being in this respect and employs ***automatic simplification*** rules for reducing expressions. Maple applies these rules when expression elements are "clearly" identical, as in many arithmetic operations:

Step 42: Automatic Simplification

```
> x:='x':                                   Ensure x is unassigned.
> x+x, x-x, x*x, x/x;          Enter a sequence of expressions. See Chapter 4 for more details.
```

$$2x,\ 0,\ x^2,\ 1$$

Maple automatically simplified each expression.

Automatic simplification also activates for "obvious" expressions, like $\sin(0)$ and $\cos(0)$. For other expressions, knowing exactly when Maple applies automatic simplification is difficult to predict. Usually, you just have to enter your input and inspect the output. Consult **?assume** for information on functions that enhance Maple's simplification capability. To apply further expression-manipulation techniques, review Chapter 7.

Practice!

33. Does Maple automatically simplify expressions, such as $\sin(x + x)$?

34. Enter $\frac{2(x+y)}{4(x+y)}$ such that Maple performs automatic simplification.

35. Does Maple simplify $\sqrt{x^2}$ to x? Why or why not? How might you use **assume** to generate x from $\sqrt{x^2}$?

3.11.1 Full Evaluation

When you enter an input, Maple performs ***full evaluation*** on all input expressions. Full evaluation is a process that

- Converts expressions into trees
- Replaces all assigned names with their respective expressions, which are also converted into trees
- Attempts to perform all possible automatic simplifications
- Performs all mathematical operations for each subexpression

To fully evaluate an expression, Maple usually starts from the bottom of the entire tree and works up until reaching the top of the tree. Maple will perform all operations and functions on all subexpressions to calculate the final result.

For example, suppose you entered the expression `sqrt(1+1/10)`. Maple would convert the expression into a tree, as shown in Figure 3-1. To fully evaluate the input, Maple performs these operations in succession: divide 1 by 10, add the result to 1, then take the square root of the entire result.

3.11.2 Assignments

How does Maple treat assignments when fully evaluating an expression? Consider these two Maple statements:

- expression ***expr***: Maple fully evaluates ***expr***, but performs no assignment.
- assignment ***name := expr***: Maple fully evaluates ***expr*** and stores the results in ***name***, if ***name*** is assignable.

In either case, Maple reports the result as output if you terminated ***expr*** with a semicolon (`;`).

What happens when Maple encounters a name *inside* ***expr***? For example, you might enter `x:=y+z`. Maple needs to check whether, or not, the names y and z have assigned values to evaluate $y + z$ and store the result in x. If a name that belongs to ***expr*** is

- assigned, Maple replaces the name with the assigned expression retrieved from memory. Maple evaluates ***expr*** with the substituted expression tree.
- unassigned, the name remains a symbolic value which Maple includes in the evaluation of ***expr***.

At no point, however, does Maple change any assignment for a name that belongs to the expression tree. Continuing the example, suppose had you entered the following input:

```
> restart: x:=y+z: y:=1: x;
```

In order of input statements, Maple first erases all assignments and assigns to x the expression $y + z$. Since y and z have no assigned values, Maple stores $y + z$ in memory for the name x. Then, the value 1 is assigned to y, but Maple does not evaluate x, yet. *Entering* **x** causes Maple to fully evaluate x by retrieving the expression $y + z$ from memory, replacing y with 1, adding 1 to z, and reporting the evaluation $1 + z$ as output.

There are more nuances…what if you change the assignment for y *after* entering **y:=1**? What if y were assigned *before* entering **x:=y+z**? Or, what if you change y and re-evaluate x? The next two sections demonstrate these implications of assignments.

3.11.3 Example 1: Assigned Names

This example demonstrates what happens when Maple encounters an assigned name. Consider the example of a sequence of Maple inputs and outputs shown in Table 3-6. The “Expression” columns indicate how Maple stores each expression in memory after the input statements are entered. Now, trace the session:

- Input ① restores variables to just their names.
- Input ② first assigns 2 to m. Then, when assigning mx to y, Maple replaces m with 2 during the evaluation. Consequently, y gets $2x$, and not mx, in memory!
- Input ③ changes m. But, y cannot change because y has $2x$ stored in memory.
- Input ④ resets m. Again, y does not change unless it gets another assignment.

Table 3-6 Full Evaluation (Assigned Names)

Order	Input Statements	Output	m	x	y
①	`> restart: y;`	y	m	x	y
②	`> m:=2: y:=m*x;`	$y := 2x$	2	x	$2x$
③	`> m:=3: y;`	$2x$	3	x	$2x$
④	`> m:='m': y;`	$2x$	m	x	$2x$

Do you see the implications of assigning a variable, like *m* in Table 3-6, before using it another expression? During full evaluation, Maple replaces all assigned names with their associated expressions. So, the resulting expression ($2x$) no longer contains those assigned names (*m*) and Maple "forgets" that you used the names previously.

3.11.4 Example 2: Unassigned Names

This example demonstrates what happens when Maple evaluates an expression that contains an unassigned name. Consider the example of a sequence of Maple inputs and outputs shown in Table 3-7. Now, trace the session:

- Input ① restores variables to just their names.
- Input ② assigns to y the expression mx.
- Input ③ gives m a value of 2. During evaluation of y, Maple replaces m with 2 and reports the result $2x$. But, Maple does not assign the result of $2x$ to y, which remains as mx in memory.
- Input ④ gives m a new value of 3 which Maple uses during evaluation of y. But, y remains assigned to mx in memory.
- Input ⑤ unassigns m. Again, the expression for y remains mx.

Table 3-7 Full Evaluation (Unassigned Names)

Order	Input Statements	Output	m	x	y
①	`> restart: y;`	y	m	x	y
②	`> y:=m*x;`	$y := mx$	m	x	mx
③	`> m:=2: y;`	$2x$	2	x	mx
④	`> m:=3: y;`	$3x$	3	x	mx
⑤	`> m:='m': y;`	mx	m	x	mx

Maple fully evaluates new values of y for inputs ② through ⑤, but the assignment for y never changes after input ②. Why? Inside Mape's memory, the name y remains assigned to the expression mx throughout future inputs until you assign another expression, unassign y, or restart your worksheet. In this fashion, Maple mirrors the human ability to "remember" an expression. This capability allows you to test different values in a previously assigned expression without having to reassign that expression.

Practice!

Predict the output from the following input statements and then test your prediction with Maple:

36. `> restart: A:=1: B; B:=A: B; A:=2: B;`

37. `> restart: B:=A: B; A:=1: B; A:=2: B;`

38. `> restart: x:=y: y:=x: y; x;`

3.11.1 Execution Groups

Maple evaluates input inside an execution group from left to right, and then, from top to bottom. So, the top input line is evaluated from left to right. The next input line is evaluated from left to right, and so forth. Note that pressing ↵ anywhere inside an execution group enters the *entire* group as input, starting at the top of the group.

3.11.2 Unevaluation

To delay full evaluation, surround an expression with forward quotes, such as **`'expr'`**. What does it mean to "delay" Maple? Delaying evaluation is called ***unevaluation***, which is a process that involves two steps after you enter **`'expr'`**:

- stripping the input **`'expr'`** of the forward quotes (**`''`**)
- attempting automatic simplification on **`expr`**

No further evaluation occurs on **`expr`**! For instance, try delaying the evaluation of **`sqrt(4)`**:

Step 43: Unevaluation

```
> sqrt(4);
```

$$2$$

First, evaluate $\sqrt{4}$ without delay!

Without quotes, Maple fully evaluates the expression.

```
> expr:='sqrt(4)';
```

$$expr := \sqrt{4}$$

Now, delay the evaluation of $\sqrt{4}$.

Maple stripped `sqrt(4)` *of its quotes..*

```
> expr;
```

$$2$$

Now, fully evaluate $expr$.

Without quotes, Maple fully evaluates $expr$.

Now do you see why entering **`name:='name'`** unassigns a name? Maple first strips **`'name'`** of the forward quotes, which leaves **`name:=name`** as input. Hence, **`name`** is assigned to

itself using the literal symbol for ***name*** without any value.Consult **?uneval** and **?eval** for more information.

Practice!

39. How does Maple compute $\frac{2^2}{4}$? What answer do you obtain?

40. Unassign the variable **x** without using **restart**.

41. Does Maple simplify the input **'1+2'**? Hint: Consult **?uneval**.

Professional Success: Managing Assignments

Have you found how Maple "remembers" assignments a bit confusing? Worksheets show only input and output without an indication of *when* computations were entered. So, you need to keep track of statement order! The following example demonstrates the danger of forgetting statement order.

Initiate a small Maple session and create four empty execution groups. Next, *type* these input lines without entering the statements:

```
> A := 1:     #1
> A := 'A':   #2
> B := A:     #3
> B;          #4
```

What input order will assign to *B* the value 1? By moving your cursor up and down, enter statements in the order #2, #3, #1, and #4.

Such situations might arise when you forget your statement entry order. Since Maple stores the most recent assignment, unexpected results often arise when one does not keep track of this order. If Maple reports bizarre results, check for missing assignments or unassignments. If you still lose track of assignments, try these tips:

- Select Edit→Execute→Worksheet to enter all input statements from top to bottom.
- Enter `anames()` to see a list of all currently assigned variables.
- Frequently check the current values of your names.
- Unassign names before assigning new expressions.
- Look for case-sensitive names and missing operators.
- Use the parallel-kernel mode.
- Delete everything, and enter `restart`, only as a last result, of course.

3.12 Application: Engineering Economics

This section demonstrates how Maple and its language elements assist solving an engineering-economics problem.

3.12.1 Background

Engineers must often choose alternative plans based on economic decisions. In this section, you calculate an item's *annual worth* (AW), which is a "leveled" annual payment that represents the item's yearly income and cost. AW converts single payments, like P the present purchase price (P) and future salvage value (SV), into an equivalent annual amount (A). Assuming a yearly interest rate i over an item's life cycle n, calculate annual worth is calculated as

$$AW = (P)(A/P, i, n) + (SV)(A/F, i, n) + A, \tag{3-1}$$

where

$$(A/P, i, n) = \frac{(i)(1+i)^n}{(1+i)^n - 1} \tag{3-2}$$

and

$$(A/F, i, n) = \frac{i}{(1+i)^n - 1}. \tag{3-3}$$

- The factor $(A/P, i, n)$ converts a present value P into an annual cash flow A.
- The factor $(A/F, i, n)$ converts an future value F into an annual cash flow A.
- The lone A in Eq. 3-1 indicates annual operating costs that do not need conversion.

3.12.2 Problem

Determine the annual worth of a machine that has a purchase price of $72000, an annual maintenance and labor cost of $1000, and a salvage value of $14000, given an interest rate 12% over a 10-year life cycle. Calculate the percent difference of annual worth for a purchase price of $65000. Percent difference is defined as

$$percent\ \ difference = \frac{|original - changed|}{original} \times 100 \tag{3-4}$$

3.12.3 Methodology

First, distinguish the cash-flow values between negative cost and positive income. Use an execution group to label important variables and parameters as text:

Step 44: Economics – Restate

```
> restart:
```

Refresh your worksheet.

Outflow: *P* (initial price) = -72000, *A* (annual cost) = -1000
Inflow: *SV* (salvage value) = +14000
Parameters: *i* (interest rate) = 12%, *n* (life cycle) = 10 years

Before calculating specific quantities, assign the formulas from Eqs. 3-1 and 3-2 in symbolic form. Because of Maple's full evaluation rules, you will be able to assign different parameters without changing how Maple stored the formulas:

Step 45: Economics – Model

```
> AgivenP:=(i*(1+i)^n)/((1+i)^n-1):          Assign (A/P,i,n).
> AgivenF:=(i)/((1+i)^n-1):                  Assign (A/F,i,n).
> 'Annual Worth' := (P)*AgivenP + (SV)*AgivenF + A;   Assign Eq. 3-1.
```

$$Annual\ \ Worth := \frac{Pi(1+i)^n}{(1+i)^n - 1} + \frac{SVi}{(1+i)^n - 1} + A$$ *Check your formula!*

Now, assign the cash-flow values:

Step 46: Economics – Separate

```
> P:=-72000: A:=-1000: SV:=14000:            Assign cash flows.
> i:=0.12: n:=10:                            Enter i as a decimal percent.
```

You can now evaluate the annual worth simply by entering the variable name. Full evaluation will take care of variable substitution because you made your assignments after assigning the formula for annual worth.

Step 47: Economics – Solve and Report

```
> AW1:='Annual Worth';                       Evaluate the annual worth.
```

$$AW1 := -13663.08199$$ *Store the result in AW1.*

To produce only a certain amount of decimal places, enter **evalf(*expr*,*d*)**, where ***d*** is the number of digits you want Maple to use in a calculation:

```
> evalf(%,7);                                Evaluate the previous expression with 7 digits.
```

$$-13663.08$$ *You could also enter* `evalf(AW1,7)`.

To compute the percent difference for a different purchase price,

assign P a new value and re-enter **'Annual Worth'**:

```
> A:=-65000: AW2:='Annual Worth';
```

$$AW2 := -12424.19284$$

Evaluate the annual worth.

Store the result in $AW2$.

Use **abs(*expr*)** to take the absolute value of ***expr***:

```
> pdiff := abs( (AW1-AW2)/AW1)*100 ):
> evalf(pdiff,4);
```

$$9.078$$

Percent difference $= \frac{|old-new|}{old} \times 100$.

Round the result.

The answer is about 9%.

3.12.4 Solution

The annual worth of the machine is $-13663.08. The percent difference in annual worth from lowering the purchase price to $65000 is about 9%. If you're curious about Maple's some built-in functions, check out **?finance**.

Summary

- The Maple language constructs expressions with tokens, which include names, operators, reserved words, integers, punctuation and strings.
- Operator precedence and associativity determine the order that Maple chooses operators when evaluating expressions.
- Maple names act as variables to which you may assign expressions.
- Reserved words and protected names are tokens that Maple has already defined and should be avoided for assignments.
- Commands for Maple to act upon, like expressions and assignments, are called statements.
- Maple converts an expression into an expression tree that divides the expression into individual tokens.
- An assignment statement evaluates an expression and stores the result in an assignable name for later use.
- Maple evaluates expressions using full evaluation and automatic simplification.
- Automatic simplification is Maple's built-in methods of calculating "obvious" expressions.
- Full evaluation decomposes an expression into a tree and evaluates each subexpression.
- Full evaluation leaves unassigned names as symbolic values and replaces assigned names with their respective expressions.
- To unassign a name, enter `name := 'name'`, which works because of unevaluation due to the forward quotes.

Key Terms

automatic simplification

dittos

expression

expression tree

full evaluation

integer

language

name

operand

operator

operator associativity

operator precedence

parallel-kernel mode

protected names

punctuation

reserved word

semantics

shared-kernel mode

statements

string

symbol

syntax

tokens

unassigning

unevaluation

variable

Problems

1. Name four elements of the Maple command language.
2. Explain the difference between a mathematical expression and a Maple expression that you enter as a statement.
3. How do Maple names and symbols differ?
4. How do protected names and reserved words differ?
5. Explain how Maple elements make up expressions.
6. Give an example of an expression that employs addition and division with integers.
7. Assume you discover the expression $[x(y+z)]$ inside another textbook. Enter the expression in Maple as input using Maple Notation. Should you avoid square brackets? Why or why not?
8. Produce the Maple output *‘I am a string that includes backquotes‘* using a Maple name as input. Hint: Consult **?name**.
9. Draw an expression tree for the expression $\sin(a+b)$. (Hint: Do this by hand!)
10. Explain the difference between shared- and parallel-kernel modes.
11. Demonstrate Maple input that will produce the following output.

11a. A 11g. $x+\frac{y}{z}$

11b.	$A + 1$	11h.	$\frac{x}{z} + y$
11c.	$A + \frac{1}{a}$	11i.	$A := x + 1$
11d.	A^2	11j.	$A \le B$
11e.	$A^3 + \frac{1}{2}$	11k.	$A = B$
11f.	$\frac{x + y}{z}$	11l.	$\sqrt{\sin(x)}$

12. Evaluate the following expressions using Maple. Hints: Be careful with operator precedence and associativity. Do not worry if Maple produces a fractional result.

12a.	$1 + 2 + 3$	12g.	$\left(\frac{1}{2}\right)\left(-\frac{3}{4}\right)$
12b.	$\frac{1}{2} + 3$	12h.	$2\left(\frac{3}{1 + 4}\right)$
12c.	$1 - 2 - 3$	12i.	2^3
12d.	$1 - (2 + 3)$	12j.	$2^{(-1)}$
12e.	$1 \times 2 \times 3$	12k.	$2^{(2-4)}$
12f.	$2 \div 3$	12l.	$2^{\frac{-1}{5 + \frac{4}{3}}}$

13. Can you assign to the name A the value 1 with the input statement `1:=A`? Why or why not? Demonstrate with Maple.

14. Perform the following tasks:

14a. Assign to the name A the value of 1.

14b. Assign to the name a the value of 2.

14c. Check the values of both names A and a.

14d. Unassign the name A.

14e. Unassign the name a.

15. Enter the following input statements:

```
> restart: y=mx+b: m=1: b=2: x=3: y;
```

16. Why does Maple *not* produce the output of 5? Correct the input statements so that Maple does. Show the corrected input with its corresponding output.

17. Assign to a the value 10. Next, store the value of a in b. Now, change the value of a to 20. Is the current value of b 10 or 20? Explain and demonstrate your answer with Maple.

18. Assign to C the value 123. Next, assign to c the same value. Are the values of C and c the same? Why or why not? Demonstrate your results and conclusions with Maple.

19. Fill in the missing information inside Table 3-8. Hint: Refer to Tables 3-6 and 3-7.

Table 3-8 Full Evaluation (Problem 19)

Order	Input Statements	Output	m	x
①	`> x:='x': y:='y':`			
②	`> x:=10;`			
③	`> y:=x+5;`			
④	`> x:=20: y;`			
⑤	`> x:='x': y;`			

20. Ceramics mix metallic and non-metallic compounds and have found wide-ranging use in many branches of technology. A type of ceramic called *spinel* has the chemical compound $MgAl_2O_4$. Express the spinel compound using Maple names for each individual element. Hints: Use Maple input. For names with an index, use square brackets, as discussed in `?name`. For instance, enter `B[1]` to express B_1.

21. If you invert the factor $(A/F, i, n)$ will you determine the factor

$$(F/A, i\%, n) = \frac{(1+i)^n - 1}{i}?$$

Why or why not? Demonstrate with Maple.

22. Many ingredients, such as cement, water, sand, and coarse aggregate (rocks), compose standard structural concrete. Assuming the proper vibration to remove entrapped air, you can approximate the total volume of concrete for a pour using the following densities: cement ρ_c = 195 pcf (pound-mass per cubic foot), water ρ_w = 62.4 pcf, and aggregate (includes rocks and sand) ρ_a = 165 pcf. Suppose that you can mix a batch of concrete that contains 250 lbm (pound-mass) of gravel, 150 lbm of sand, 100 lbm of cement, and 50 lbm of water. How many batches of concrete do you need to fill a rectangular wall with dimensions 20 ft × 5 ft × 1 ft? Assume no volume loss due to rebar reinforcement. Hint: Both gravel and sand make up aggregate, so add their weights together.

22a. Restart Maple.

22b. Assign the densities. For instance, `rho[w]:=62.4*l6m/ft^3`.

22c. Calculate the volume of each component: Volume = Density (lbm/ft3) × Mass (lbm).

22d. Calculate the total volume in one batch. Hint: Sum the volumes in the previous step.

22e. Calculate the required volume of concrete, i.e., the volume of the wall.

22f. Calculate the number of batches: Batches = Volume of wall (ft^3) ÷ Volume of Batch $(\text{ft}^3/\text{batch})$.

23. Assume an elastic bar is axially loaded with a force P, as shown in Figure 3-9.

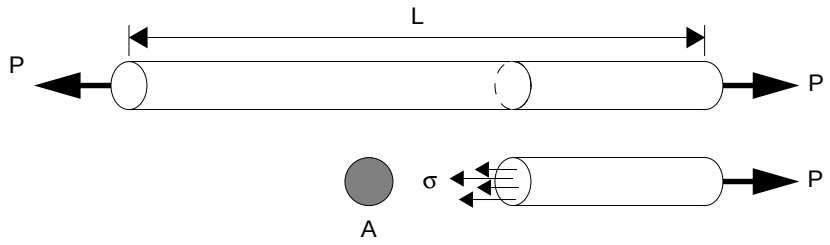

Table 3-9 Elastic Axial Bar

Engineering *strain* ε is defined as a body's change of length Δ divided by the original length L. The engineering stress σ on the bar uniformly divides the load P by the original area A before deformation. Thus,

$$\varepsilon = \frac{\Delta}{L} \text{ and } \sigma = \frac{P}{A}. \tag{3-5}$$

Assuming a linear relationship $\sigma = E\varepsilon$ between stress and strain, solve these problems:

23a. Assign to the stress σ the expression $E\varepsilon$. E is called Young's Modulus. Hint: Refer to Appendix A and the symbol palette for entering Greek names.

23b. Assuming fundamental units of force and length, what units do stress, strain, and Young's Modulus have? Hint: Consider the formulas for ε and σ.

23c. Plot stress versus strain for Young's Modulus $E = 10$ for $0 \le \varepsilon \le 1$.

23d. Substitute the expression for stress P/A and that for strain Δ/L into the relationship $\sigma = E\varepsilon$. You can do this by hand, but type the results as text in Maple. You should determine that $P/A = E\Delta/L$.

23e. Assign to the name EQN the entire expression determined in the above problem. Your assignment should have the form **`EQN:=expr1=expr2`**. Hints: Include the equal sign (**`=`**)! This expression is called an *equation*.

23f. Solve for P using Maple. Hints: Investigate **`?solve`**. Your input should have the form **`solve(EQN,something)`**. Your output should be $E\Delta A/L$.

23g. Hooke's Law for a spring states that $P = K\Delta$. Relate this equation to your results in the above step. You should be able to show a formula for K in terms of E, A, and L. Compare and contrast the equations.