

5.1 Computational Toolbox—Tools of the Trade: *Mathematica* Tutorial

2

File: *MathematicaTutorial2.nb*

Introduction to Computational Science: Modeling and Simulation for the Sciences
Angela B. Shiflet and George W. Shiflet
Wofford College
© 2006 by Princeton University Press

Introduction

The prerequisite to this tutorial is "*Mathematica* Tutorial 1." Tutorial 2 prepares you to use *Mathematica* to complete projects for this and subsequent chapters. The tutorial introduces the following functions and concepts: *List*, *ListPlot*, *Joined*, comments, and *Append*. The module also gives examples along with Quick Review Questions for you to do with *Mathematica*. Execute all input cells to view the results of the examples.

Lists

Many *Mathematica* applications involve **lists**, and a number of built-in functions perform operations on lists. **Braces**, **{ }**, enclose the values in a list, such as follows:

```
numList = {13, 36, 92}
```

We also employ a list to store an ordered pair, such as the following representation of the point (-3, 46):

```
orderedPair = {-3, 46}
```

- **Quick Review Question 1** Do anything that is asked of you in cells that look like this one, marked as a Quick Review Question in boldface. Because such cells are text cells and not input cells, do not type in these cells. Instead, move the cursor down until it changes from being vertical to being a horizontal-I bar. Click and start typing; a new input cell will form.

For this question, assign to *ptLst* a list representing the following ordered pairs (points): (-3, 6), (5, 2), and (1, 12).

The elements of a list have a numeric order starting with 1, and we can refer to a particular element by using the name of the list and double square brackets, `[[]]`, surrounding that number. For example, the second element of *numList* above is as follows:

```
numList[[2]]
```

Besides using a number, such as 2, we can employ a variable, such as *i*, that has a value, so that *numList[[i]]* refers to the *i*-th element of the list. Consequently, a *Do* loop can process the elements of a list individually by having the varying loop index specify the list element.

■ **Quick Review Question 2** Using *Print* and a *Do* loop, print on separate lines the points of *ptLst* from Quick Review Question 1.

Frequently, a list occurs within a list, such as the following list that represents a matrix with two rows, each in braces, and four columns:

```
mat = {{45, 99, 203, -29}, {775, 31, -582, 62}}
```

With *mat* having two elements, the rows, we reference the first row, {45, 99, 203, -29}, as follows:

```
mat[[1]][[3]]
```

Alternatively, we employ the following notation that separates the indices with a comma:

```
mat[[1, 3]]
```

- **Quick Review Question 3** Write commands using *ptLst* and double square brackets to return the following parts of *ptLst* from Quick Review Question 1:
 - a. The third point, (1, 12)
 - b. The first coordinate of the second point, 5. Use two pairs of double square brackets.
 - c. The first coordinate of the second point, 5. Use one pair of double square brackets.
 - d. The second coordinate of the first point, 6. Use two pairs of double square brackets.
 - e. The second coordinate of the first point, 6. Use one pair of double square brackets.

Graphing Points

To plot points in a list, we use *ListPlot*. The segment below assigns a list of points to the variable *pts* and then displays the points with the appropriate axes labels by employing the option *AxesLabel*, which is also available for *Plot*. In this example, two statements appear in one cell, the assignment of a list to *pts* and *ListPlot*. As the following segment indicates, output for each command occurs after the input cell:

```
pts = {{-3, 2}, {2, 2}, {-1, -1}, {4, 3}, {0, 0}}
ListPlot[pts, AxesLabel → {" x ", " y "}]
```

- **Quick Review Question 4** Graph the points in list *ptLst* from the previous Quick Review Question. Have labeled axes.

In the command below, we make the points bigger and assign the graph to a variable, *lp*, for later reuse. To have larger points, we employ the option *PlotStyle* with the directive *PointSize*, which indicates the diameter of points as a fraction of the graph's width. Thus, the following command designates that the diameter of any point be $0.03 = 3\%$ of the total width of the graphics:

```
lp = ListPlot[pts, AxesLabel → {" x ", " y "},
  PlotStyle → {PointSize[0.03]}]
```

- **Quick Review Question 5** Copy to a new cell the answer of the previous Quick Review Question to plot the list of points stored in variable *ptLst*. Adjust the command to have the points appear larger.

Lines Connecting Points

Sometimes, it is helpful to visualize the path of an entity, such as an animal or a molecule, whose movement we are simulating. To generate the path, we specify `True` for the `ListPlot` option `Joined`, as `Joined -> True`. The subsequent graph displays line segments joining pairs of adjacent points.

For example, suppose `{-1, 0, -1, 0, 1, 2, 3, 2, 1, 0}` is a list (*ylst*) of *y* values, where each element is randomly one more or less than the previous element. Considering each *y* value to occur at sequential ticks of the clock, we can draw a line graph to display the trend of the *y* values over time. Just plotting *ylst* causes the first coordinates of the points to be 1, 2, ..., 10 and the corresponding second coordinates to be from *ylst*. The segment to display the trend of *y* over time follows:

```
ylst = {-1, 0, -1, 0, 1, 2, 3, 2, 1, 0};
```

```
lp = ListPlot[ylst, AxesLabel -> {"t", "y"}, Joined -> True]
```

- **Quick Review Question 6** Plot the list of points stored in variable *ptLst* from the Quick Review Questions of the "Lists" section. Label the axes, and connect the points with line segments.

Comments

Any program, regardless of the language, should have ample comments to explain the code. It is amazingly easy forget what was done only a few minutes earlier. It takes far less time to enter the comments as we type the program than to figure out the code later. A descriptive software engineering phrase is "Write once, read many times."

We have been using *Mathematica* notebook cells with styles, such as *Text*, to record comments. However, for longer segments of code, internal comments are also helpful. In *Mathematica*, comments appear between `(*` and `*)`, and the computer ignores text between this pair of delimiters. Examples of comments follow:

```
(* A comment can appear on a line or lines by itself
or can document code on a line *)
lst = {}; (* initialize list to empty *)
```

- **Quick Review Question 7** Write a statement to assign 80 to the variable *v*, and in a comment on the same line indicate that the variable represents "velocity in km/hr."

Appending

In simulations with *Mathematica*, we frequently build a list, one element at a time. To do so, we employ the *Mathematica* function `Append`, which returns a list with an additional element appended on the end. The form of a call to the function, where *expr* is the list and *elem* is the additional element, follows:

```
Append[expr, elem]
```

The segment below returns the value of the list *lst* with a new element, 5, on the end. As the output of *lst* on the last line indicates, however, `Append` returns the appended list but does not change the original value of *lst*.

```
lst = {1, 2, 3, 4};
Append[lst, 5]
lst
```

To have *lst* be the appended list, we assign the value of the `Append` function to *lst*, as follows:

```
lst = Append[lst, 5]
lst
```

The appended element can be a list itself. In the following example, *pts* originally represents a list of two points. After execution, *pts* contains a third point, the origin.

```
pts = {{1, 5}, {-2, 7}};
pts = Append[pts, {0, 0}]
```

In a programming language, such as C, C++, or Java, when using a loop to count, we must initialize the counter to be zero before the loop. Similarly, we must initialize the first argument of `Append` to be a list before invocation of `Append`. When building a list from scratch, that initial value is usually an **empty list**, `{}`. The segment below defines a function $g(x) = 3\sqrt{x}$ and then stores $g(i)$ for the positive integers i less than 10 in the list *gLst*, which is originally empty. Finally, we have *Mathematica* return the value of *gLst*.

```
Clear[g]
g[x_] := 3  $\sqrt{x}$ 

gLst = {};
Do[
  gLst = Append[gLst, g[i]],
  {i, 9}
]

gLst
```

- **Quick Review Question 8** Using the comments in the cell below as a guide, write a *Mathematica* segment to generate a list, *lst*, of 30 points. Initialize the list to be empty and *y* to be 1. Inside the body of a *Do* loop with an integer index *i* that goes from 0 through 29, make *x* be 0.25 times *i*, make *y* be 1.2 times its previous value, and append the ordered pair of values to the list. After creation of the list of points, plot the points with line segments joining adjacent points and labeled axes.

```
(* initialize list lst to be empty *)
(* initialize y to be 1 *)
(* for i going from 0 through 29 in a loop do the following: *)
(*   assign to x the expression 0.25 times i *)
(*   assign to y the expression 1.2 times previous value of y *)
(*   assign to lst the list with ordered pair of x and y appended *)

(* plot the points *)
```