"When I look at my bookshelf, I see eleven books on Perl programming. *Perl by Example, Third Edition*, isn't on the shelf; it sits on my desk, where I use it almost daily. I still think it is the best Perl book on the market for anyone—beginner or seasoned programmer—who uses Perl daily."

—Bill Maples, Enterprise Network Support, Fidelity National Information Services

PERL by EXAMPLE

FIFTH EDITION

Ellie Quigley

Praise for Ellie Quigley's Books

"I picked up a copy of *JavaScript by Example* over the weekend and wanted to thank you for putting out a book that makes JavaScript easy to understand. I've been a developer for several years now and JS has always been the 'monster under the bed,' so to speak. Your book has answered a lot of questions I've had about the inner workings of JS but was afraid to ask. Now all I need is a book that covers Ajax and Coldfusion. Thanks again for putting together an outstanding book."

—Chris Gomez, Web services manager,
Zunch Worldwide, Inc.

"I have been reading your *UNIX® Shells by Example* book, and I must say, it is brilliant. Most other books do not cover all the shells, and when you have to constantly work in an organization that uses tcsh, bash, and korn, it can become very difficult. However, your book has been indispensable to me in learning the various shells and the differences between them…so I thought I'd email you, just to let you know what a great job you have done!"

—Farogh-Ahmed Usmani, B.Sc. (Honors), M.Sc., DIC,
project consultant (Billing Solutions), Comverse

"I have been learning Perl for about two months now; I have a little shell scripting experience but that is it. I first started with *Learning Perl* by O'Reilly. Good book but lacking on the examples. I then went to *Programming Perl* by Larry Wall, a great book for intermediate to advanced, didn't help me much beginning Perl. I then picked up *Perl by Example, Third Edition*—this book is a superb, well-written programming book. I have read many computer books and this definitely ranks in the top two, in my opinion. The examples are excellent. The author shows you the code, the output of each line, and then explains each line in every example."

—Dan Patterson, software engineer,
GuideWorks, LLC

"Ellie Quigley has written an outstanding introduction to Perl, which I used to learn the language from scratch. All one has to do is work through her examples, putz around with them, and before long, you're relatively proficient at using the language. Even though I've graduated to using *Programming Perl* by Wall et al., I still find Quigley's book a most useful reference."

—Casey Machula, support systems analyst,
Northern Arizona University, College of Health and Human Services

EXAMPLE 8.21 (CONTINUED)

```
(Output)
Christian Dave Dan Christian
Betty Christian Dick Christian
Igor Norma Christian Christian
```

EXPLANATION

2 With the *g* option, the substitution is global. **Every** occurrence of *Tom* will be replaced with *Christian* for each line that is read.

3 The *DATA* filehandle gets its data from the lines that follow the _ _*DATA*_ _ token.

The *i* Modifier—Case Insensitivity

Perl is sensitive to upper- or lowercase characters when performing matches. If you want to turn off case sensitivity, an *i* (insensitive) is appended to the last delimiter of the match or substitution operator and the search pattern will be case insensitive, whereas this has no effect on the replacement side.

FORMAT

```
s/search pattern/replacement string/i;
```

EXAMPLE 8.22

```
(The Script)
   # Matching with the i option
   use warnings;
1  while(<DATA>){
2     print if /norma cord/i;    # Turn off case sensitivity
   }
3  __DATA__
   Steve Blenheim
   Betty Boop
   Igor Chevsky
   Norma Cord
   Jon DeLoach
   Karen Evich

(Output)
Norma Cord
```

EXPLANATION

2 Without the *i* option, the regular expression */norma cord/* would not be matched, because all the letters are not lowercase in the lines that are read as input. The *i* option turns off case sensitivity.

3 The *DATA* filehandle gets its data from the lines that follow the _ _*DATA*_ _ token.

EXAMPLE 8.23

```
(The Script)
   use warnings;
1  while(<DATA>){
2     print if s/igor/Daniel/i;   # Substitute igor with Daniel
   }

3  __DATA__
   Steve Blenheim
   Betty Boop
   Igor Chevsky
   Norma Cord
   Jon DeLoach
   Karen Evich

(Output)
Daniel Chevsky
```

EXPLANATION

2 The regular expression in the substitution is also case insensitive, owing to the *i* option. If *igor* or *Igor* (or any combination of upper- and lowercase) is matched, it will be replaced with *Daniel*.

3 The *DATA* filehandle gets its data from the lines that follow the *__DATA__* token.

The *e* Modifier—Evaluating an Expression

On the replacement side of a substitution operation, it is possible to evaluate an expression or a function. The search side is replaced with the result of the evaluation.

FORMAT

```
s/search pattern/replacement string/e;
```

EXAMPLE 8.24

```
(The Script)
   use warnings;
   # The e and g modifiers
1  while(<DATA>){
2     s/6/6 * 7.3/eg;       # Substitute 6 with product of 6 * 7.3

3     print;

   }
   __DATA__
   Steve Blenheim   5
   Betty Boop       4
   Igor Chevsky     6
   Norma Cord       1
   Jon DeLoach      3
   Karen Evich      66
```

EXAMPLE 8.24 (CONTINUED)

```
(Output)
Steve Blenheim      5
Betty Boop          4
Igor Chevsky        43.8
Norma Cord          1
Jon DeLoach         3
Karen Evich         43.843.8
```

EXPLANATION

2 If the *$_* scalar contains the number *6*, the replacement side of the substitution is evaluated. In other words, the *6* is multiplied by *7.3* (*e* modifier); the product of the multiplication (*43.8*) replaces the number *6* each time the number *6* is found (*g* modifier).

3 Each line is printed. The last line contained two occurrences of *6*, causing each *6* to be replaced with *43.8*.

EXAMPLE 8.25

```
(The Script)
   use warnings;
   # The e modifier
1  my $number = 5;
2  $number =~ s/5/6 * 4 - 22/e;
3  print "The result is: $number\n";

4  $number = 51055;
5  $number =~ s/5/3 * 2/eg;
6  print "The result is: $number\n";

(Output)
3  The result is: 2
6  The result is: 61066
```

EXPLANATION

1 The *$number* scalar is assigned *5*.

2 The *s* operator searches for the regular expression *5* in *$number*. The *e* modifier evaluates the replacement string as a numeric expression and replaces it with the result of the arithmetic operation, *6* 4 - 22*, that is, *2*.

2 *$number* is assigned *1055*.

5 The *s* operator searches for the regular expression *5* in *$number*. The *e* modifier evaluates the replacement string as a numeric expression and replaces it with the product of *3 * 2*; that is, every time *5* is found, it is replaced with *6*. Since the substitution is global, all occurrences of *5* are replaced with *6*.

EXAMPLE 8.26

```
(The Script)
   use warnings;
1  my $line = "knock at heaven's door.\n";
2  $line =~ s/knock/"knock, " x  2  . "knocking"/ei;
3  print "He's $line;

(Output)
He's knock, knock, knocking at heaven's door.
```

EXPLANATION

1 The *$line* variable is the string *"knock at heaven's door.\n"*;
2 The *s* operator searches for the regular expression *knock* in *$line*. The *e* modifier evaluates the replacement string as a string expression and replaces it with *knock x 2* (repeated twice) and concatenates (the dot operator) with the string *knocking*, ignoring case.
3 The resulting string is printed.

Using the Special *$&* Variable in a Substitution. The special *$&* variable is used to hold the pattern that is found on the search side of a substitution. Its value is used in the replacement side when performing an evaluation, but it is a read-only variable, meaning you cannot change it; for example, you cannot use *$&* += 5.

EXAMPLE 8.27

```
(The Script)
   use warnings;
1  my $salary=50000;
2  $salary =~ s/$salary/$& * 1.1/e;
3  print "\$& is $&\n";
4  print "The salary is now \$$salary.\n";

5  my $line = "knock at heaven's door.\n";
6  $line =~ s/knock/"$&, " x  2  . "knocking"/ei;
7  print "He's $line";

(Output)
3  $& is 50000
4  The salary is now $55000.
6  He's knock, knock, knocking at heaven's door.
```

EXPLANATION

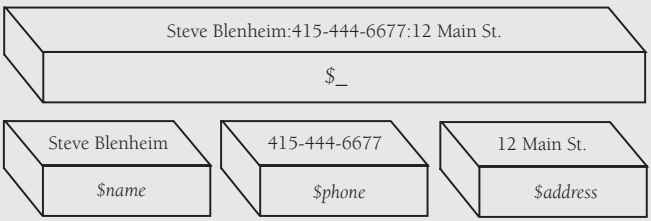
1 The scalar *$salary* is assigned *50000*.
2 The substitution is performed on *$salary*. The replacement side evaluates the expression. The special variable *$&* holds the value found on the search side. To change the value in *$salary* after the substitution, the pattern matching operator =~ is used. This binds the result of the substitution to the scalar $*salary*.

EXPLANATION (CONTINUED)

3 The *$&* scalar holds the value of what was found on the search side of the substitution.
4 The scalar *$salary* has been increased by *10%*.
5 The *$line* scalar is assigned the string *"knock at heaven's door.\n"*.
6 If the search string (*knock*) is found, it is stored in the *$&* variable. On the replacement side, the expression is evaluated. So, the value of *$&* (*knock*) is replicated twice and concatenated with *$&ing* (*knocking*). The new value is substituted for the original value. *$line* is assigned the new value and printed.

Pattern Matching with a Real File. In all the previous examples, we have been using the *DATA* filehandle for performing pattern matches and substitutions with regular expressions. The following examples demonstrate how you can use pattern matching when working with lines from an external file.

EXAMPLE 8.28




```
(The Script)
   # Using split, an anonymous list, and pattern matching
   my($name,$phone,$address);
1  open(my $fh, "<", "datafile") or die "Can't open file $!";
2  while(<$fh>){
3     ($name, $phone, $address) = split(":", $_);
4      print "$name\n" if $phone =~ /408-/   # Using the pattern
                                             # matching operator
   }
5  close $fh

(The file)
Steve Blenheim:415-444-6677:12 Main St.
Betty Boop:303-223-1234:234 Ethan Ln.
Igor Chevsky:408-567-4444:3456 Mary Way
Norma Cord:555-234-5764:18880 Fiftieth St.
Jon DeLoach:201-444-6556:54 Penny Ln.
Karen Evich:306-333-7654:123 4th Ave.

(Output)
Igor Chevsky
```

EXPLANATION

1 The user-defined lexical variable *$fh* is associated with the external file, *datafile*. It is opened for reading.

3 The *split* function will split each line, *$_*, as it is read from the file and return a list consisting of three scalars: *$name*, *$phone*, and *$address*.

4 The pattern */408-/* is matched against the *$phone* variable. If that pattern is matched, the value of *$name* is printed. *Igor*'s name is printed because his phone matches the *408* area code. In the next chapter, we will use regular expression metacharacters to control how the pattern is matched; for example, *$phone =~ /^408/* will match only if the phone begins with 408. The ^ is the beginning-of-line anchor.

5 The file is closed. The contents of the file that is being used for this example are listed.

EXAMPLE 8.29

```
(The Script)
   # use warnings;
   my($name,$phone,$address);
1  open(my $fh, "<", "datafile") or die "Can't open datafile";
2  while(my $inputline=<$fh>){
3     ($name, $phone, $address) = split(":", $inputline);
4     print $inputline if $name =~ /^Karen/;
5     print if /^Norma/;  # $_ is empty
   }
(The datafile)
Steve Blenheim:415-444-6677:12 Main St.
Betty Boop:303-223-1234:234 Ethan Ln.
Igor Chevsky:408-567-4444:3456 Mary Way
Norma Cord:555-234-5764:18880 Fiftieth St.
Jon DeLoach:201-444-6556:54 Penny Ln.
Karen Evich:306-333-7654:123 4th Ave.

(Output)
4  Karen Evich:306-333-7654:123 4th Ave.
5  < No output >
```

EXPLANATION

2 Each line is read from *FH* and stored in *$inputline*, one after the other, until the end of the file is reached.

3 *$inputfile* will be split at the colons and the value returned stored in an anonymous list consisting of three scalars: *$name*, *$phone*, and *$address*.

4 The value of *$inputline* is displayed if the value of *$name* begins with *Karen*.

5 Since *$_* is not being used in this example to hold the current line, nothing will print here. The line reads: print the value of *$_* if *$_* contains the pattern *Norma*.