```
#!/usr/bin/perl
# example of all kinds of perl features, last revised 25 November 2013
# modify that top line to point to a particular version of perl
print "Hello, world!\n";  # a short Perl example

# on UNIX, use the chmod command to make this file executable, e.g.
# chmod +x PerlCheatSheet.pl
# then just run this file like any executable, e.g. ./PerlCheatSheet
# no .pl suffix is needed since the shell will find it
# or invoke perl explicitly, e.g.
# this example causes Perl to run the program with warnings turned on
# perl -w -Mdiagnostics PerlCheatSheet.pl

use strict;  # an example of a pragma
use warnings;                                   A

# Perl statements end with a semicolon
# ordinary variables begin with $
my $nDocs = 0;

# perl is great for working with strings
# strings are delimited with either single quotes or double quotes
my $aString = 'This string extends over two lines, and most escapes like \n
have no effect';
my $bString = "Inside double quotes, usual escapes apply";  # and interpolation
# period . is the string concatenation operator
my $aLongString = $aString."\nconcatenated with the period operator\n".
"and the x factor for repetition\n$bString\nand interpolation!";
print $aLongString x 2;
# note use of . as part of assignment, like several ops in C
$bString .= "\n";

# LOTS of pattern matching operators, including s for substitute
$aString = "Do not do that!\n";
print $aString;
$aString =~ s/Do not/Don't/;
print $aString;                                 B

# usual operators, with associativity and precedence as in C
# exceptions being comaprison ops for strings
my $fred = "fred\n";
my $barney = "barney\n";
print "is fred lt barney?\n";
# nothing special about Boolean variables
# if it's zero, it's false, otherwise it's true
my $aBoolean = $fred lt $barney;

# control structures include if
# although we just defined $aBoolean, we can always test
if (defined($aBoolean) && $aBoolean) {
    print "fred is less than barney\n";
} else {
    print "fred is NOT less than barney\n";    C
}

#control structures include while, for, and foreach
my $count = 0;
while ($count < 10) {
    $count += 2;
    print "count is now $count\n";
}

# Perl supports lists and arrays, closely related concepts
# but there's no notion of type
# subscripts start at zero normally, as in C
my @fred;
$fred[0] = 2.8;
$fred[1] = "Wilma!";
$fred[2] = 'dino' x 2;
my @barney = ("this is", "also a list but with ", 4, "elements");      D

# negative subscripts (!) count from the end of the array
print "last SUBSCRIPT of array fred is $#fred\n";
print "but the last element of array fred is $fred[-1]\n";

# push and pop add or delete elements from the end of a list
my @fs = qw/fred wilma barney betty/;
push @fs, qw/bambam pebbles/;
# shift and unshift delete or elements from the start of a list
unshift @fs, "dino";
# list elements are separated by blanks when interpolated
print "@fs\n";
my @sf = reverse(@fs);
printf "Print the list in reverse @sf\n";

# run an external command and save the output                   E
# to read all the input from STDIN
#my @lines = <STDIN>
# from Learning Perl by Brian d foy
my @lines = `perldoc -u -f atan2`;
# sometimes we consider what Perl expects, i.e. list vs. scalar context
#my $nLinesRead = @lines;
my $nLinesRead = scalar @lines;   # the keyword scalar forces scalar context
print "Value of nLinesRead is $nLinesRead\n";
# to read a single line of input
#my $textLine = "a line of text\n";
#chomp($textLine = <STDIN>);

# make this handout
my $thisFile = $0;
my $handout = $thisFile;                                        F
$handout =~ s/pl$/pdf/;
print "Creating a handout called $handout using $thisFile\n";
my $rc = `enscript -2r -p - -M Letter $thisFile | ps2pdf - $handout`;

# invoke a subroutine
my $nRowsWritten = &xlsDemo("PCSlexicon.xls");
print "Wrote a spreadsheet with $nRowsWritten row(s) written.\n";

# working with files and directories, and patterns
my @files = <*>;                                                G
foreach my $myFile (@files) {
    if (-f $myFile and -r $myFile) {   # see if it's a readable file
        my ($nl, $nw, $nch) =
            `wc $myFile` =~ /([0-9]+)\s+([0-9]+)\s+([0-9]+)/;
        print "file $myFile has $nl lines, $nw words and $nch characters\n";
    }
}

# what's the weather like?
use Weather::Airport;
my $wa = Weather::Airport->new;                                 H
my $airport = $wa->query('BWI');
use Data::Dumper;
print Dumper ($airport);

# spreadsheet demo
sub xlsDemo {
    # note use of @_ in both the scalar and list contexts in this if
    my $XLSfile = "default.xls";
    if (@_ == 1) {
        # list of arguments in list context
        ($XLSfile) = @_;
    } else {
        print "usage: echoDemo(inputFile=STDIN[,XLSfile=default.xls]\n";
    }
```

```
    my $word;
    my $nTerms=0;

    # Perl has built-in hash functions
    my %aHashTable = ();  # will be used in example below
    print "The name of the file now running is $0\n";
    open THISFILE, $0;

    while (my $inputLine = <THISFILE>) {
        chomp $inputLine;  # get rid of trailing newline
        #print STDOUT "$inputLine\n";   #  I/O looks very C-like, eh?

        # recall that array names begin with @
        # split $inputLine into an array of blank-separated words

        my @words=split(" ", $inputLine);
        foreach $word (@words) {
            $aHashTable{$word} += 1;

            # to practice with regular expression matching, see if word
            # would be a good password, i.e. having at least one
            # digit, one lower case letter, and one upper case letter

            # make sure it finds a good password when run on itself XYzzy18
            if ($word =~ /[A-Z]/ && $word =~ /[a-z]/ && $word =~ /[0-9]/) {
                print "$word would be a good password.\n";
            }
        }
    }

    for $word (sort(keys(%aHashTable))) {
        $nTerms++;
        # but let's make each word lower-case
        # and make a variable wordlc local to this block
        my $wordlc = $word;
        $wordlc =~ tr/A-Z/a-z/;  # tr stands for translate
        #printf STDOUT "%s, %d\n", $wordlc, $aHashTable{$word};
    }

# let's make a spreadsheet
    use Spreadsheet::WriteExcel;
# make some new objects
# the my keyword makes them local to this sub
    my $workbook = Spreadsheet::WriteExcel->new($XLSfile);
    my $worksheet = $workbook->add_worksheet();
# write two column headers
    $worksheet->write(0,0,"term");
    $worksheet->write(0,1,"count");

# now write each term, and its count, on its own row
    my $row = 1;
    foreach my $term (sort(keys(%aHashTable))) {
        $worksheet->write($row, 0, sprintf("\"%s\"", $term));
        $worksheet->write($row, 1, $aHashTable{$term});
        $row++;
    }
    print "Should have written $row rows in the spreadsheet\n";
    return $row;
}
```

J

K

L

```
bash-3.2$ ./PerlCheatSheet.pl
Hello, world!
... edited output a lot to fit on page!...
last SUBSCRIPT of array fred is 2
but the last element of array fred is dinodino
dino fred wilma barney betty bambam pebbles
Print the list in reverse pebbles bambam betty barney wilma fred dino
Value of nLinesRead is 16
Creating a handout called ./PerlCheatSheet.pdf using ./PerlCheatSheet.pl
[ 2 pages * 1 copy ] left in -
The name of the file now running is ./PerlCheatSheet.pl
$nTerms=0; would be a good password.
XYzzy18 would be a good password.
Should have written 520 rows in the spreadsheet
Wrote a spreadsheet with 520 row(s) written.
...
file sum1toN.c has 42 lines, 155 words and 921 characters
file TWFschedule.html has 632 lines, 1978 words and 32974 characters
file wc.hs has 12 lines, 56 words and 403 characters
$VAR1 = [
        ' Temperature: 33° F (0° C) ',
        ' Dewpoint: Unknown ',
        ' Barometer: 30.29 ',
        ' Sea Level Pressure: 30.292 ',
        ' Trend: Increasing ',
        ' Change: 50.0% ',
        ' Updated: 17:00 UTC ',
        ' Date: Tue Nov-26-2013 ',
        ' Includes the Counties: Anne Arundel ',
        ' Includes the Cities: Annapolis '
      ];
bash-3.2$
```