

Source file: example

```
#####  
#  
# A simple example of a Maple 'source' file. Such a file can  
# be easily created and maintained using your favorite text  
# editor and can contain arbitrary Maple commands.  
# I find this mechanism particularly useful for developing  
# and maintaining Maple procedures.  
#  
# The file can most easily be read into a Maple session by  
# first 'cd'-ing to the directory which contains this file:  
#  
# % cd /Public/Members/matt/Src/maple/examples  
#  
# starting up maple or xmaple  
#  
# % maple  
#  
# or  
#  
# % xmaple &  
#  
# then typing  
#  
# > read example;  
#  
# If maple (or xmaple) isn't running in the directory  
# containing this file, then you must use an absolute  
# pathname and be sure to enclose the name in backquotes  
# or double-quotes. (This also applies to filenames  
# containing a '.', which is why I tend to use simple  
# names (no extensions) for files containing Maple source.)  
#  
# > read '/Public/Members/matt/Src/maple/examples/example';  
#  
# Recall that use of the colon (:) as terminator rather  
# than semi-colon (;) inhibits echoing of results.  
#  
#####  
  
aa := 23 / 155;  
  
myadd := proc(x::numeric, y::numeric)  
    x + y;  
end;
```

Source file: ladd

```
#####
# ladd: Adds all elements of a list.
#####
ladd := proc(l::list)

#-----
# Define local variables.
#-----
    local lsum, i:

#-----
# Check for valid argument, exit with error message
# if not valid.
#-----
    if nops(l) = 0 then
        ERROR('argument is the NULL list');
    fi;

#-----
# Initialize sum to first element of list.
#-----
    lsum := l[1];

#-----
# Loop over rest of elements accumulating the sum.
#-----
    for i from 2 to nops(l) do
        lsum := lsum + l[i];
    od;

#-----
# Return the sum.
#-----
    lsum;
end:

#####
# ladd: Alternative, more compact implementation using
# 'add' procedure. Not possible before Maple V.4.
#####
laddnew := proc(l::list)
    local i;
    add( l[i], i=1..nops(l) );
end:
```

Source file: t1add

```
#####  
# Tests  
#   ladd  
#   laddnew  
#####  
read ladd;  
  
l1 := [1,2,3,4];  
  
#####  
# Note the use of the 'printf' procedure, and the '%a'  
# format specification.  
#####  
printf("ladd(...) is %a \n", ladd(l1));  
printf("laddnew(...) is %a \n", laddnew(l1));
```

Source file: tladd-trace

Script started on Tue Sep 21 15:48:45 2004

```
lnx1 1> pwd
/home/phys410/maple
```

```
lnx1 2> cat tladd
```

```
#####
# Tests
#   ladd
#   laddnew
#####
read ladd;

l1 := [1,2,3,4];

#####
# Note the use of the 'printf' procedure, and the '%a'
# format specification.
#####
printf("ladd(...) is %a \n", ladd(l1));
printf("laddnew(...) is %a \n", laddnew(l1));
```

```
lnx1 3> maple
```

```
 | \ ^ / |      Maple 8 (IBM INTEL LINUX)
._ | \ |   | / | _ . Copyright (c) 2002 by Waterloo Maple Inc.
 \  MAPLE / All rights reserved. Maple is a registered trademark of
 < ____ ____ > Waterloo Maple Inc.
   |          Type ? for help.
> read tladd;
```

```
l1 := [1, 2, 3, 4]
```

```
ladd(...) is 10
laddnew(...) is 10
> quit;
bytes used=142088, alloc=196572, time=0.05
lnx1 4> exit
exit
```

Script done on Tue Sep 21 15:49:03 2004

Source file: polyinterp

```
#####
#
# polyinterp: Constructs Lagrange Interpolating Polynomial
#
# Given n distinct "data points" (x_i,f_i) , i = 1 ... n, and a name,
# this procedure returns the unique polynomial (in name) of degree
# n - 1 which passes through (interpolates) all the points.
#
# Input parameters:
#
#   ldata:   list of lists, which defines (x_i,f_i)
#   var:     name, returned interpolating polynomial is
#            a polynomial in 'var'
#
# Usage example:
#
#   > polyinterp([ [0,1], [1,6], [2,4], [3,0] ], 'x' );
#
#               3      2
#           5/6 x  - 6 x  + 61/6 x + 1
#
# Implementation notes:
#
#   This routine converts the list of input pairs (each pair
#   itself a two-element list) to separate *sequences* of
#   the x_i and f_i.  You could also build up separate *lists*
#   but it is syntactically easier to build sequences in Maple.
#
#####
polyinterp := proc(ldata::list(list),var::name)
#-----
#   Local variables:
#
#   n:           number of data points
#   i,  j:      loop variables used in evaluation of Lagrange formula
#   sx, sf:     for building up sequences of x_i, f_i respectively
#   num, den:   for building up the numerators and denominators of the
#               characteristic polynomials.
#   p:          for building up the interpolating polynomial itself
#
#-----
  local n,  i,  j,  sx,  sf,  num,  den,  p;

#-----
#   Determine number of data points
#-----
  n := nops(ldata);
```

```

#-----
# Initialize polynomial and x_i and f_i sequences
#-----
p := 0;
sx := NULL;
sf := NULL;
#-----
# Convert input list-of-lists into separate sequences of x_i and f_i
#-----
for i from 1 to n do;
    sx := sx , ldata[i][1];
    sf := sf , ldata[i][2];
od;

#-----
# For each of the x_i ...
#-----
for i from 1 to n do;
#-----
# ... build up the numerators and denominators of the ith
# characteristic polynomial. First initialize the numerator
# and denominator ...
#-----
num := 1;
den := 1;
#-----
# ... and then build them up using the Lagrange formula. Note that
# both the numerator and denominator are products of n - 1
# terms, one term for each j = 1..n such that j <> i.
#-----
for j from 1 to n do;
    if j <> i then
        num := num * (var - sx[j]);
        den := den * (sx[i] - sx[j]);
    fi
od;
#-----
# Update the polynomial
#-----
p := p + sf[i] * (num / den);
od;
#-----
# Return the polynomial in expanded form
#-----
expand(p);

end:

```