```
> 2+3^4;
```

$$83 \tag{1}$$

```
> x+x;
```

$$2\,x \tag{2}$$

```
> expand((x+4)^2);
```

$$x^2 + 8\,x + 16 \tag{3}$$

```
> x := 4;
```

$$x := 4 \tag{4}$$

```
> x;
```

$$4 \tag{5}$$

```
> x+x = 2*x;
```

$$8 = 8 \tag{6}$$

```
> {1,1,2,3,5} intersect {5,6,7};
```

$$\{5\} \tag{7}$$

```
> li := [1,1,2,3,5];
```

$$li := [1, 1, 2, 3, 5] \tag{8}$$

```
> li[4];
```

$$3 \tag{9}$$

```
> sin(Pi);
```

$$0 \tag{10}$$

```
> restart;
> li;
```

$$li \tag{11}$$

```
> x := 4;
```

$$x := 4 \tag{12}$$

```
> x := 'x';
```

$$x := x \tag{13}$$

```
> x;
```

$$x \tag{14}$$

```
> alias(s=sin(x),c=cos(x));
```

$$s, c \tag{15}$$

```
> sin(x)*cos(x);
```

$$s\,c \tag{16}$$

```
> alias(J=BesselJ);
```

$$s, c, J \tag{17}$$

```
> diff(BesselJ(n,sin(x)),x,x,x);
```

$$\left( -\left( \left( -J(n+1,s) + \frac{n\,J(n,s)}{s} \right) c + \frac{(n+1)\,J(n+1,s)\,c}{s^2} \right. \right. \tag{18}$$

$$-\frac{(n+1)\left(J(n,s)-\frac{(n+1)\,J(n+1,s)}{s}\right)c}{s}\Bigg)c+\Bigg(J(n,s)$$

$$-\frac{(n+1)\,J(n+1,s)}{s}\Bigg)s+\frac{2\,n\,J(n,s)\,c^2}{s^3}-\frac{2\,n\left(-J(n+1,s)+\frac{n\,J(n,s)}{s}\right)c^2}{s^2}$$

$$+\frac{n\,J(n,s)}{s}+\frac{1}{s}\Bigg(n\Bigg(-\left(J(n,s)-\frac{(n+1)\,J(n+1,s)}{s}\right)c-\frac{n\,J(n,s)\,c}{s^2}$$

$$+\frac{n\left(-J(n+1,s)+\frac{n\,J(n,s)}{s}\right)c}{s}\Bigg)c\Bigg)-n\left(-J(n+1,s)+\frac{n\,J(n,s)}{s}\right)\Bigg)c-2\Bigg($$

$$-\left(J(n,s)-\frac{(n+1)\,J(n+1,s)}{s}\right)c-\frac{n\,J(n,s)\,c}{s^2}$$

$$+\frac{n\left(-J(n+1,s)+\frac{n\,J(n,s)}{s}\right)c}{s}\Bigg)s-\left(-J(n+1,s)+\frac{n\,J(n,s)}{s}\right)c$$

```
> restart;
> f := x -> sin(x);
```

$$f := x \to \sin(x) \tag{19}$$

```
> f(Pi);
```

$$0 \tag{20}$$

```
> diff(f(z),z$4);
```

$$\sin(z) \tag{21}$$

```
> z$4;
```

$$z, z, z, z \tag{22}$$

```
> li := [seq(i^2,i=1..4)];
```

$$li := [1, 4, 9, 16] \tag{23}$$

```
> sum(i, i=0..n);
```

$$\frac{1}{2}(n+1)^2-\frac{1}{2}n-\frac{1}{2} \tag{24}$$

```
> add(i, i=0..100);
```

$$5050 \tag{25}$$

```
> add(i, i in li);
```

$$30 \tag{26}$$

```
> Sum(i, i=0..n);
```

$$\sum_{i=0}^{n} i \tag{27}$$

```
> int(sin(x), x);
```

$$-\cos(x) \tag{28}$$

```
> Int(sin(x), x=0..Pi);
```

$$\int_0^{\pi} \sin(x)\, dx \tag{29}$$

```
> evalb(evalf(sqrt(5) < 2));
```

*false* (30)

```
> evalb(8=8);
```

*true* (31)

```
> eval(sin(x)/x,x=0);
Error, numeric exception: division by zero
> subs(x=0,sin(x)/x);
Error, numeric exception: division by zero
> ?index,packages
> with(LinearAlgebra);
```

[ *&x*, *Add*, *Adjoint*, *BackwardSubstitute*, *BandMatrix*, *Basis*, *BezoutMatrix*, *BidiagonalForm*, *BilinearForm*, *CharacteristicMatrix*, *CharacteristicPolynomial*, *Column*, *ColumnDimension*, *ColumnOperation*, *ColumnSpace*, *CompanionMatrix*, *ConditionNumber*, *ConstantMatrix*, *ConstantVector*, *Copy*, *CreatePermutation*, *CrossProduct*, *DeleteColumn*, *DeleteRow*, *Determinant*, *Diagonal*, *DiagonalMatrix*, *Dimension*, *Dimensions*, *DotProduct*, *EigenConditionNumbers*, *Eigenvalues*, *Eigenvectors*, *Equal*, *ForwardSubstitute*, *FrobeniusForm*, *GaussianElimination*, *GenerateEquations*, *GenerateMatrix*, *Generic*, *GetResultDataType*, *GetResultShape*, *GivensRotationMatrix*, *GramSchmidt*, *HankelMatrix*, *HermiteForm*, *HermitianTranspose*, *HessenbergForm*, *HilbertMatrix*, *HouseholderMatrix*, *IdentityMatrix*, *IntersectionBasis*, *IsDefinite*, *IsOrthogonal*, *IsSimilar*, *IsUnitary*, *JordanBlockMatrix*, *JordanForm*, *LA_Main*, *LUDecomposition*, *LeastSquares*, *LinearSolve*, *Map*, *Map2*, *MatrixAdd*, *MatrixExponential*, *MatrixFunction*, *MatrixInverse*, *MatrixMatrixMultiply*, *MatrixNorm*, *MatrixPower*, *MatrixScalarMultiply*, *MatrixVectorMultiply*, *MinimalPolynomial*, *Minor*, *Modular*, *Multiply*, *NoUserValue*, *Norm*, *Normalize*, *NullSpace*, *OuterProductMatrix*, *Permanent*, *Pivot*, *PopovForm*, *QRDecomposition*, *RandomMatrix*, *RandomVector*, *Rank*, *RationalCanonicalForm*, *ReducedRowEchelonForm*, *Row*, *RowDimension*, *RowOperation*, *RowSpace*, *ScalarMatrix*, *ScalarMultiply*, *ScalarVector*, *SchurForm*, *SingularValues*, *SmithForm*, *StronglyConnectedBlocks*, *SubMatrix*, *SubVector*, *SumBasis*, *SylvesterMatrix*, *ToeplitzMatrix*, *Trace*, *Transpose*, *TridiagonalForm*, *UnitVector*, *VandermondeMatrix*, *VectorAdd*, *VectorAngle*, *VectorMatrixMultiply*, *VectorNorm*, *VectorScalarMultiply*, *ZeroMatrix*, *ZeroVector*, *Zip* ] (32)

```
> LinearAlgebra[RandomMatrix](2);
```

$$\begin{bmatrix} 44 & -31 \\ 92 & 67 \end{bmatrix} \tag{33}$$

```
> A := Matrix([[2,3,4],[3,5,7],[4,5,8]]);
```

$$A := \begin{bmatrix} 2 & 3 & 4 \\ 3 & 5 & 7 \\ 4 & 5 & 8 \end{bmatrix} \quad (34)$$

```
> b := Vector([4,8,6]);
```

$$b := \begin{bmatrix} 4 \\ 8 \\ 6 \end{bmatrix} \quad (35)$$

```
> A.b;
```

$$\begin{bmatrix} 56 \\ 94 \\ 104 \end{bmatrix} \quad (36)$$

```
> <A|b>;
```

$$\begin{bmatrix} 2 & 3 & 4 & 4 \\ 3 & 5 & 7 & 8 \\ 4 & 5 & 8 & 6 \end{bmatrix} \quad (37)$$

```
> for x by 2 from 1 while x<6 to 10 do print(x); od;
```

```
1
3
5
```

(38)

```
> x := 4;
```

$$x := 4 \quad (39)$$

```
> x := "string";
```

$$x := \text{"string"} \quad (40)$$

```
> fib := proc(n::nonnegint)
    option remember;
    local x,y,z;
    if n<2 then
      return(n);
    else
      return(fib(n-1)+fib(n-2));
    fi;
  end:

> [seq(fib(i), i=0..10)];
```

$$[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55] \quad (41)$$

```
> fib(50);
```

$$12586269025 \quad (42)$$

```
> fib(z);
Error, (in fib) cannot determine if this expression is true or
false: z < 2
```

```
> fib(-1);
Error, invalid input: fib expects its 1st argument, n, to be of
type nonnegint, but received -1
> interface(verboseproc=3);
                                  1                                  (43)
> print(factor);
proc(xFP, K)                                                         (44)
    option remember, system,
    Copyright (c) 1991 by the University of Waterloo. All rights reserved.;
    local x, z, u, f, r, num, den, myfactor, subsIndexed;
    if type(xFP, {'list, set, relation, series, range'}) then return map(factor, args) end if;
    if not type(xFP, 'algebraic') then return xFP end if;
    if hastype(xFP, 'float') then
        x := evalf(xFP);
        if nargs = 2 and not member(args[2], ['real', 'complex']) then
            error
            "floating point coefficients are incompatible with field extension; use 'real' or
            'complex' instead"
        end if
    else
        x := xFP
    end if;
    if member(_Z, indets(x)) then x := subs(_Z=z, x) end if;
    subsIndexed := map(y→y=tools/gensym('_factor'), indets(x, 'indexed'));
    myfactor := proc(y) if y=x then y else factor(y) end if end proc;
    if nargs = 2 then
        if K = [ ] or K = { } then
            if type(x, 'ratpoly(rational)') then
                r := factor(x)
            elif type(x, 'algfun'('algnum')) then
                r := factor(x, evala/GetAlgExt(x))
            elif type(x, 'radfun'('radalgnum')) then
                r := convert(factor(convert(simplify(x, 'radical'), 'RootOf'), K), radical)
            else
                error "expecting a polynomial over an algebraic number field"
            end if
        elif type(K, {'algnumext', 'list'('algnumext'), 'set'('algnumext')}) then
            if type(x, 'polynom(algnum)') then
                r := normal(evala('Factor'(x, K)))
            elif type(x, 'algfun'('algnum')) then
                r := normal(factor/algext(x, K))
            elif type(x, 'algfun') then
```

```
            r := frontend(myfactor, subs(subsIndexed, [x, K]), [{`*`, `^`, `=`, `+`, 'set',
            'list', 'RootOf', 'complex'('extended_numeric')}, {}])
        else
            error "expecting a polynomial over an algebraic number field"
        end if
    elif type(K, {'radnumext', 'list'('radnumext'), 'set'('radnumext')}) then
        if type(x, 'polynom(radnum)') then
            u := radfield(union(indets(x, radnum), indets(K, radnum)));
            r := subs(u[2], factor(subs(u[1], x), u[3]))
        elif type(x, 'radfun'('radalgnum')) then
            f := factor(convert(x, 'RootOf'), convert(K, 'RootOf')); r := convert(f, 'radical
            ')
        elif type(x, {'algfun'('algnum'), 'polynom'('algnum')}) then
            r := factor(x, convert(K, 'RootOf'))
        elif type(x, 'radfun') then
            r := frontend(myfactor, subs(subsIndexed, [x, K]), [{`*`, `^`, `=`, `+`, 'set',
            'list', 'RootOf', 'complex'('extended_numeric')}, {}])
        else
            error "expecting a polynomial over a radical number field"
        end if
    elif type(K, {'algext'('algfun'('algnum')), 'list'('algext'('algfun'('algnum')))), 'set'(
    'algext'('algfun'('algnum'))))}) then
        if type(x, 'algfun'('algnum')) then
            r := normal( factor/algext(x, K))
        else
            error "expecting a polynomial over an algebraic function field"
        end if
    elif type(K, {'radext'('radfun'('algnum')), 'list'('radext'('radfun'('algnum')))), 'set'(
    'radext'('radfun'('algnum'))))}) then
        if type(x, 'radfun'('radalgnum')) then
            f := factor(convert(x, 'RootOf'), convert(K, 'RootOf')); r := convert(f, 'radical
            ')
        else
            error "expecting a polynomial over an algebraic function field"
        end if
    elif K = 'real' or K = 'complex' then
        r := evalf/Factor(x, K)
    else
        error "2nd argument, %1, is not a valid algebraic extension", K
    end if
elif nargs <> 1 then
```

```
        error "wrong number of arguments"
    elif not hastype(x, ``+``) then
        r := x
    elif type(x, 'polynom'('rational')) then
        r := factor/factor(x)
    elif type(x, 'polynom'('complex'('rational'))) then
        r := factor(x, I)
    elif type(x, 'polynom'('complex'('numeric'))) then
        r := evalf/Factor(x)
    elif type(x, 'ratpoly'('numeric')) then
        r := normal(x);
        num := numer(r);
        den := denom(r);
        f := factor(num), factor(den);
        if f[1] <> num or f[2] <> den then r := normal(f[1]/f[2]) end if
    elif type(x, 'anything'^Not('integer')) then
        r := map(factor, x)
    elif type(x, 'polynom'('algnum')) then
        r := normal(evala('Factor'(x)))
    elif type(x, 'ratpoly'('algnum')) then
        r := factor(x, evala/GetAlgExt(x))
    elif type(x, {'polynom'('radnum'), 'ratpoly'('radnum')}) then
        u := radfield(indets(x, 'radnum')); r := subs(u[2], factor(subs(u[1], x)))
    elif type(x, 'function') then
        r := tools/map('factor', x)
    elif assign('u', normal(x)) = NULL and u <> xFP then
        r := factor(u)
    elif type(x, {'`+`, `*`, `^`'}) then
        f := map(y -> y = factor(y), indets(x));
        r := frontend(myfactor, [subs(union(f, subsIndexed), x)], [{`*`, `=`, `+`, 'set', 'list',
        'complex'('extended_numeric')}, {}])
    else
        r := x
    end if;
    r := subs(map(rhs = lhs, subsIndexed), r);
    subs(z = _Z, r)
end proc
```

```
> interface(verboseproc=3);
```



(45)

```
> print(diff);
```

```
proc( )
```

(46)

```
    option builtin = diff, remember;

end proc#(sin(x), x) = cos(x)#(sin(z), z) = cos(z)#(cos(z), z) = `+`(`-`(sin(z)))#(sin(x), x) = cos
    (x)
```

```
> op(1..3,a+b+c);
```

$$a, b, c \tag{47}$$

```
> dismantle[hex](x-2*y^3);

SUM(1C5D08CC,5)
   NAME(815115C,4): x
   INTPOS(3,2): 1
   PROD(1C5D7FF4,3)
      NAME(81C955C,4): y
      INTPOS(7,2): 3
   INTNEG(FFFFFFFD,2): -2
```