```
// Borwein kvadratikus algoritmusa

N:=5:
DIGITS:=73:

a:=sqrt(2): b:=0: p:=2+sqrt(2):
for i from 1 to N do
 uj_a:=(sqrt(a)+1/sqrt(a))/2:
 uj_b:=(sqrt(a)*(1+b))/(a+b):
 uj_p:=p*uj_b*(1+uj_a)/(1+uj_b):
 a:=uj_a:
 b:=uj_b:
 p:=uj_p:
 print(float(p)):
 print(Unquoted, "Eltérés: ".expr2text(float(p-PI))):
 end_for;
```

```
// Borwein kvadratikus algoritmusa

N:=6:
DIGITS:=500:

a:=sqrt(2): b:=0: p:=2+sqrt(2):
for i from 1 to N do
 uj_a:=(sqrt(a)+1/sqrt(a))/2:
 uj_b:=(sqrt(a)*(1+b))/(a+b):
 uj_p:=p*uj_b*(1+uj_a)/(1+uj_b):
 a:=uj_a:
 b:=uj_b:
 p:=uj_p:

 elteres:=float(p-PI):
 tizedesjegy_pontossag:=trunc(abs(trunc(log(10,elteres)))):
 print(Unquoted, "Eltérés: ".
  expr2text(tizedesjegy_pontossag).
  " tizedesjegy."):
 end_for;
```

```
// Borwein kvartikus algoritmusa

N:=5:
DIGITS:=1000:

a:=6-4*sqrt(2): y:=sqrt(2)-1:
for i from 0 to N-1 do
 uj_y:=(1-(1-y^4)^(1/4))/(1+(1-y^4)^(1/4)):
 uj_a:=a*((1+uj_y)^4)-2^(2*i+3)*uj_y*(1+uj_y+uj_y^2):
 y:=uj_y:
 a:=uj_a:
 p:=1/a:

 elteres:=float(p-PI):
 tizedesjegy_pontossag:=trunc(abs(trunc(log(10,elteres)))):
 print(Unquoted, "Eltérés: ".
  expr2text(tizedesjegy_pontossag).
  " tizedesjegy."):
 end_for;
```

```
// Ooura kvadratikus algoritmusa a PI kiszámítására.
// A módszer a Gauss-Legendre-féle számtani-mértani közép (AGM) gyorsított
// változata; az eredeti programban a szorzásokat gyors
// Fourier-transzformációval (FFT) végzik (ami további gyorsítás).
// Copyright (C) 1999 Takuya Ooura <oouraemmm.t.u-tokyo.ac.jp>
// http://momonga.t.u-tokyo.ac.jp/~ooura/fft.html

DIGITS:=1000:                            // Itt kell megadni a pontosságot.
ELTERES:=10^(-DIGITS):                   //
SQRT_SQRT_ELTERES:=sqrt(sqrt(ELTERES)):  //
n:=1:                                    //

    c := sqrt(0.125):
    a := 1 + 3 * c:
    b := sqrt(a):
    e := b - 0.625:
    b := 2 * b:
    c := e - c:
    a := a + e:
    npow := 4:
    while e > SQRT_SQRT_ELTERES do
        npow := 2 * npow:
        e := (a + b) / 2:
        b := sqrt(a * b):
        e := e - b:
        b := 2 * b:
        c := c - e:
        a := e + b:
        n := n + 1:                      //
        end_while:
    e := e * e / 4:
    a := a + b:
    p := (a * a - e - e / 2) / (a * c - e) / npow:

float(PI);                               //
print(Unquoted, "Eltérés: ".             //
 expr2text(float(p-PI)));                //
print(Unquoted, "Lépésszám: ".           //
 expr2text(n));                          //
```

```
/*++
A1_I_40_1 -- Az Analízis I. példatár I/40. feladatából az 1. megoldása

A1_I_40_1 (x,n)

x - bemenő x (a feladatban 2)
n - bemenő n
++*/

A1_I_40_1 := proc (x,n)
begin
    if n=1 then sqrt(x)
        else sqrt(x*A1_I_40_1(x,n-1));
        end_if;
    end_proc:

// Példa:
A1_I_40_1(2, 10);
```

```
/*++
A1_I_40_3 -- Az Analízis I. példatár I/40. feladatából a 3. megoldása

A1_I_40_3 (x,n)

x - bemenő x (a feladatban c)
n - bemenő n
++*/

A1_I_40_3 := proc (x,n)
begin
    if n=1 then sqrt(x)
        else sqrt(x+A1_I_40_3(x,n-1));
        end_if;
    end_proc:

// Példa:
A1_I_40_3(2, 10);
float(%);
```

```
/*++
A1_I_41_1 -- Az Analízis I. példatár I/41. feladatából az 1. megoldása

A1_I_41_3 (x,n)

x - bemenő x
n - bemenő n
++*/

A1_I_41_1 := proc (x,n)
begin
    if n=1 then 0
        else x+A1_I_41_1(x,n-1)^2;
        end_if;
    end_proc:

// Példa:
A1_I_41_1(-2, 10);
float(%);
```

```
/*++
A1_I_30_121 -- Az Analízis I. példatár I/30. feladatából a 121. megoldása

A1_I_30_121 (x,n)

x - bemenő x
n - bemenő n
++*/

A1_I_30_121 := proc (x,n)
begin
    float(cos(x/n)^n);
    end_proc:

// Példa:
A1_I_30_121(0.2, 100);

///////////////////////////////////////////////////////////////////////////////

/*++
A1_I_30_121a -- Az Analízis I. példatár I/30./121. feladat közelítő megoldása

A1_I_30_121a (x,h,m)

x - bemenő x
h - Cauchy-féle hibaküszöb
m - maximális lépésszám
++*/

// Addig növeljük n-et, amíg a sorozat két szomszédos elemének különbsége
// nem csökken h alá. (Cauchy-féle konvergenciakritérium.)
A1_I_30_121a := proc (x,h,m)
local i,v,f,f_e,d;
begin
    if m<2 then
        return(FAIL);
        end_if;
    i:=1;
    v:=TRUE;
    while v and i<=m do
        f:=A1_I_30_121(x,i);
        if i>1 then
            d:=float(abs(f-f_e));
            if d < h then
                v:=FALSE;
                end_if; // float...
            end_if;  // i>1...
        i:=i+1;
        if v and i<=m then
            f_e:=f;
            end_if;
        end_while; // v and...
    i:=i-1;
    if not v then
        print(Unquoted,"Az n=".expr2text(i-1)." esetben |f(n+1)-f(n)|<h");
    else
        print(Unquoted,"Az n=".expr2text(i-1)." esetben |f(n+1)-f(n)|=".
            expr2text(d));
        end_if;
    print(Unquoted,"f(n)=".expr2text(f_e).", f(n+1)=".expr2text(f));
    if not v then
        return(f);
```



```
    else
        return(FAIL);
        end_if;
    end_proc:

// Példa:
A1_I_30_121a(0.5,0.0001,100);
A1_I_30_121a(0.5,0.00001,100);

///////////////////////////////////////////////////////////////////////////////

/*++
A1_I_30_121b -- Az Analízis I. példatár I/30./121. feladat közelítő megoldása

A1_I_30_121a (x,h,m)

x - bemenő x
h - Cauchy-féle hibaküszöb
m - maximális lépésszám
++*/

// Addig növeljük n-et, amíg a sorozat szomszédos elemének különbsége
// nem csökken h alá. (Cauchy-féle konvergenciakritérium.)
// Az "a" verzióhoz képest annyit módosítunk, hogy n helyébe 2^n-et írunk.
A1_I_30_121b := proc (x,h,m)
local i,v,f,f_e,d;
begin
    if m<2 then
        return(FAIL);
        end_if;
    i:=1;
    v:=TRUE;
    while v and i<=m do
        f:=A1_I_30_121(x,2^i);
        if i>1 then
            d:=float(abs(f-f_e));
            if d < h then
                v:=FALSE;
                end_if; // float...
            end_if;  // i>1...
        i:=i+1;
        if v and i<=m then
            f_e:=f;
            end_if;
        end_while; // v and...
    i:=i-1;
    if not v then
        print(Unquoted,"Az n=2^".expr2text(i-1)." esetben |f(2n)-f(n)|<h");
    else
        print(Unquoted,"Az n=2^".expr2text(i-1)." esetben |f(2n)-f(n)|=".
            expr2text(d));
        end_if;
    print(Unquoted,"f(n)=".expr2text(f_e).", f(2n)=".expr2text(f));
    if not v then
        return(f);
    else
        return(FAIL);
        end_if;
    end_proc:

// Példa:
A1_I_30_121b(0.5,0.000000000001,100);
```

```
////////////////////////////////////////////////////////////////////////////////

// Ezután számos határérték-számítási feladatot gyorsan meg tudunk oldani:
A1_I_39_7 := proc (n)
local szorzat,i;
begin
    szorzat:=1;
    for i from 1 to 2*n-1 step 2 do
        szorzat:=szorzat*i;
        end_for;
    return(n/(szorzat^(1/n)));
    end_proc:

// Lusták trükkje:
A1_I_30_121 := proc (x,n)
begin
    float(A1_I_39_7(n));
    end_proc:

// Példa (sajnos, ennél a sorozatnál problémás...):
A1_I_30_121b(0.5,0.00001,10);
```

```
/*++
A1_I_20 -- Az Analízis I. példatár I/20. feladatának megoldása

A1_I_20 (a,m,n)

a - bemenő a
m - monotonitási irány (1: növekvő, -1: csökkenő)
n - a vizsgált n-ek maximális száma
++*/

A1_I_20 := proc (a,m,n)
local v,i,f,f_e;
begin
    if abs(m) <> 1 or n=1 then
        return(FAIL);
        end_if;

    v:=TRUE;
    i:=1;
    while v and (i<=n) do
        f:=i*(a^(1/i)-1);
        print(float(f));
        if i>1 then
            /*
             * Ha itt nem konvertálunk float-ba, akkor hibaüzenetet
             * kaphatunk, ugyanis csak valós számokat lehet összehasonlítani
             * egymással:
             */
            if float(f-f_e)*m<0 then v:=FALSE;
                end_if;
            end_if;
        f_e:=f;
        i:=i+1;
        end_while;

    i:=i-1;
    if not v then
        i:=i-1;
        end_if;
    if m=-1 then
        print(Unquoted,"Monoton csökkenés n=".expr2text(i)."-ig");
        else
        print(Unquoted,"Monoton növekedés n=".expr2text(i)."-ig");
        end_if;
    end_proc:

// Példa:
A1_I_20(3.5,-1,10);
```

```
/*
 * Trapézformula, ld. Móricz Ferenc: Numerikus módszerek az algebrában
 * és az analízisben, 130. o. illetve Leindler László: Analízis, Nemzeti
 * könyvkiadó, 1993, 195. o.
 * A sum eljárás beépített függvény.
 */

Trapez := proc (N)
begin
    return((b-a)/N*(f(a)/2+f(b)/2+sum(f(a+k*(b-a)/N),k=1..N-1)));
end_proc:

factor(Trapez(6));

Simpson := proc (N)
begin
    return((b-a)/(6*N)*(f(a)+f(b)+
     4*sum(f(a+(2*k+1)*(b-a)/(2*N)),k=0..N-1)+
     2*sum(f(a+(2*k)*(b-a)/(2*N)),k=1..N-1)));
end_proc:

factor(Simpson(3));

/*
 * Romberg-rekurzió
 */

T := proc (k,m)
begin
    if m=Nil or m=0 then
        return(Trapez(k));
        end_if;
    return((4^m*T(2*k,m-1)-T(k,m-1))/(4^m-1));
end_proc:

////////////////////////////////////////////////////////////////////////////////

f:=sin:
a:=0:
b:=PI:

DIGITS:=20:


/*
 * Határozott integrál számítása (beépített függvény)
 */

int(f(x),x=a..b);

/*
 * Illusztráljuk, hogy a trapézformula mennyire lassú konvergenciát ad
 * (a számolásigény is nagy): 1000-es egyenlő beosztásnál csak 5 tizedesjegy
 * pontosságot kapunk.
 */

for i from 1 to 3 do
    print(float(Trapez(10^i)));
    end_for;

/*
 * A Simpson-formula lényegesen gyorsabban közelít.
```



```
 */

for i from 1 to 3 do
    print(float(Simpson(10^i)));
    end_for;

/*
 * A Romberg-iteráció k=1 esetén már az m=4 esetben 8 jegy pontosságot ad.
 */

for i from 1 to 4 do
    print(T(1,i));
    print(float(T(1,i)));
    end_for;

/*
 * k növelése tovább javítja a konvergencia sebességét.
 */

for i from 1 to 4 do
    print(T(2,i));
    print(float(T(2,i)));
    end_for;

/*
 * A Romberg-iteráció "elég gyors".
 */

DIGITS:=50:
for i from 1 to 10 do
    elteres:=float(2-T(1,i));
    tizedesjegy_pontossag:=trunc(abs(trunc(log(10,elteres))));
    print(Unquoted, "Pontosság: ".
     expr2text(tizedesjegy_pontossag).
     " tizedesjegy.");
    end_for;
```

```
/*
 * Ez a program szimbolikusan kiszámolja a Romberg-féle integrálásnál
 * fellépő T(k,m) kifejezéseket. Az eljárást részletesen ld.
 * Móricz Ferenc: Numerikus módszerek az algebrában és az analízisben c.
 * könyvében, 136. o.; az ott leírt eljárásnál olvasható jelölés helyett
 * itt T(k,m) az ottani T(2^m*k,m)-et jelöli.
 * Copyright (C) Kovács Zoltán, 2002/03/16
 */

T := proc (k,m)
begin
    if m=Nil or m=0 then
        return(procname(k));
        end_if;
    return((4^m*T(2*k,m-1)-T(k,m-1))/(4^m-1));
end_proc:

T(k,0);
factor(T(k,1));
factor(T(k,2));
factor(T(k,3));
factor(T(k,7));
```