lsg07

Aufgabe 1

```
reset()
```

```
P = primes_first_n(100); P
```

```
    [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
    67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
    139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
    211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277,
    281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359,
    367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439,
    443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521,
    523, 541]
```

```
L = [2*n for n in [2..400]]

# oder:
# L = range(4, 801, 2)
```

```
summen = [p1+p2 for p1 in P for p2 in P]
```

```
for x in L:
   if x not in summen:
       print "%i ist nicht Summe zweier Primzahlen <=541" % x
```

```
# oder:
all([x in summen for x in L])
```

```
    True
```

```
# oder:
Set(summen).intersection(Set(L)) == Set(L)
```

```
    True
```

```
for x in L:
   c = 0
   for s in summen:
       if s==x:
           c = c+1
   print (x,c)
```

```
WARNING: Output truncated!
full_output.txt


(4, 1)
(6, 1)
(8, 2)
(10, 3)
(12, 2)
(14, 3)
(16, 4)
(18, 4)
(20, 4)
(22, 5)
(24, 6)
(26, 5)
(28, 4)
(30, 6)
(32, 4)
(34, 7)
(36, 8)
(38, 3)
(40, 6)
(42, 8)
(44, 6)
(46, 7)
(48, 10)
(50, 8)
(52, 6)
(54, 10)
(56, 6)
(58, 7)
(60, 12)
(62, 5)
(64, 10)
(66, 12)
(68, 4)
(70, 10)
(72, 12)
(74, 9)
(76, 10)
(78, 14)
(80, 8)
(82, 9)
(84, 16)
(86, 9)
(88, 8)
(90, 18)
(92, 8)
(94, 9)
(96, 14)
```

```
(98, 6)
(100, 12)
(102, 16)
(104, 10)
(106, 11)
(108, 16)
(110, 12)
(112, 14)
(114, 20)
(116, 12)
(118, 11)
(120, 24)
(122, 7)

...

(682, 18)
(684, 28)
(686, 16)
(688, 14)
(690, 44)
(692, 12)
(694, 17)
(696, 28)
(698, 13)
(700, 24)
(702, 32)
(704, 16)
(706, 15)
(708, 20)
(710, 14)
(712, 18)
(714, 32)
(716, 14)
(718, 15)
(720, 34)
(722, 12)
(724, 10)
(726, 28)
(728, 12)
(730, 16)
(732, 32)
(734, 13)
(736, 16)
(738, 24)
(740, 14)
(742, 18)
(744, 18)
(746, 15)
(748, 16)
(750, 30)
(752, 10)
```

```
(754, 12)
(756, 26)
(758, 7)
(760, 18)
(762, 20)
(764, 12)
(766, 9)
(768, 24)
(770, 18)
(772, 16)
(774, 22)
(776, 10)
(778, 13)
(780, 30)
(782, 8)
(784, 12)
(786, 18)
(788, 8)
(790, 12)
(792, 24)
(794, 11)
(796, 8)
(798, 24)
(800, 10)
```

full_output.txt

```
# oder:
show([(x, len([s for s in summen if s==x])) for x in L])
```

$$[(4,1),(6,1),(8,2),(10,3),(12,2),(14,3),(16,4),(18,4),(20,4),$$

```
L2 = range(4,1001,2)
for x in L2:
   if x not in summen:
      print "%i ist nicht Summe zweier Primzahlen <=541" % x
```

```
968 ist nicht Summe zweier Primzahlen <=541
992 ist nicht Summe zweier Primzahlen <=541
```

```
# oder:
[x for x in L2 if x not in summen]
```

```
[968, 992]
```

```
# oder:
Set(L2).difference(Set(summen))
```

```
{968, 992}
```

Aufgabe 2

```
reset()
var('x')
```

```
    x
```

```
# hier x^0, damit substitution in der nächsten zelle funktioniert
# (x^0 ist ein symbolischer Ausdruck, 1 ein Integer)
T = [x^0, x]
for k in [2..9]:
  T.append((2*x*T[k-1]-T[k-2]).expand())
T
```

```
    [1, x, 2*x^2 - 1, 4*x^3 - 3*x, 8*x^4 - 8*x^2 + 1, 16*x^5 - 20*x^3 +
    5*x, 32*x^6 - 48*x^4 + 18*x^2 - 1, 64*x^7 - 112*x^5 + 56*x^3 - 7*x,
    128*x^8 - 256*x^6 + 160*x^4 - 32*x^2 + 1, 256*x^9 - 576*x^7 +
    432*x^5 - 120*x^3 + 9*x]
```

```
[t(x=1/3) for t in T]
```

```
    [1, 1/3, -7/9, -23/27, 17/81, 241/243, 329/729, -1511/2187,
    -5983/6561, 1633/19683]
```

```
[t(x=0.33) for t in T]
```

```
    [1, 0.330000000000000, -0.782200000000000, -0.846252000000000,
    0.223673680000000, 0.993876628800000, 0.432284895008000,
    -0.708568598094720, -0.899940169750515, 0.114608086059380]
```

```
sol = [solve(t==0,x) for t in T]; show(sol)
```

$$[], [x = 0], \left[x = -\frac{1}{2}\sqrt{2}, x = \frac{1}{2}\sqrt{2}\right], \left[x = -\frac{1}{2}\sqrt{3}, x = \frac{1}{2}\sqrt{3}, x = 0\right],$$

```
map_threaded(lambda s: s.rhs().n(), sol)
```

```
    [[], [0.000000000000000], [-0.707106781186548, 0.707106781186548],
    [-0.866025403784439, 0.866025403784439, 0.000000000000000],
    [-0.923879532511287, 0.923879532511287, -0.382683432365090,
    0.382683432365090], [-0.951056516295154, 0.951056516295154,
    -0.587785252292473, 0.587785252292473, 0.000000000000000],
    [-0.707106781186548, 0.707106781186548, -0.965925826289068,
    0.965925826289068, -0.258819045102521, 0.258819045102521],
    [-0.974927912181824 + 2.08897248911918e-17*I, 0.974927912181824 -
    2.08897248911918e-17*I, -0.433883739117558 - 2.34693928788885e-17*I,
    0.433883739117558 + 2.34693928788885e-17*I, -0.781831482468030 -
    3.90735912998779e-17*I, 0.781831482468030 + 3.90735912998779e-17*I,
    0.000000000000000], [-0.980785280403230, 0.980785280403230,
    -0.195090322016128, 0.195090322016128, -0.831469612302545,
    0.831469612302545, -0.555570233019602, 0.555570233019602],
    [-0.866025403784439, 0.866025403784439, -0.984807753012208 -
    5.59236432129384e-18*I, 0.984807753012208 + 5.59236432129384e-18*I,
    -0.342020143325669 + 8.05128564611314e-18*I, 0.342020143325669 -
    8.05128564611314e-18*I, -0.642787609686539 + 4.28399961222398e-18*I,
    0.642787609686539 - 4.28399961222398e-18*I, 0.000000000000000]]
```

```
# abschneiden des kleinen imaginärteils
map_threaded(lambda s: s.rhs().n().real(), sol)
```

```
    [[], [0.000000000000000], [-0.707106781186548, 0.707106781186548],
    [-0.866025403784439, 0.866025403784439, 0.000000000000000],
    [-0.923879532511287, 0.923879532511287, -0.382683432365090,
    0.382683432365090], [-0.951056516295154, 0.951056516295154,
    -0.587785252292473, 0.587785252292473, 0.000000000000000],
    [-0.707106781186548, 0.707106781186548, -0.965925826289068,
    0.965925826289068, -0.258819045102521, 0.258819045102521],
    [-0.974927912181824, 0.974927912181824, -0.433883739117558,
    0.433883739117558, -0.781831482468030, 0.781831482468030,
    0.000000000000000], [-0.980785280403230, 0.980785280403230,
    -0.195090322016128, 0.195090322016128, -0.831469612302545,
    0.831469612302545, -0.555570233019602, 0.555570233019602],
    [-0.866025403784439, 0.866025403784439, -0.984807753012208,
    0.984807753012208, -0.342020143325669, 0.342020143325669,
    -0.642787609686539, 0.642787609686539, 0.000000000000000]]
```

Aufgabe 3

```
reset()
var('x')
```

```
    x
```

```
f(x) = 4*x^2 + 3*x^6
g(x) = 12*x^6 - 2*x + 1
h(x) = 21*x^4 - 2*x^2 + 12
```

```
k(x) = f(g(g)); k
```

```
    x |--> 3*(12*(12*x^6 - 2*x + 1)^6 - 24*x^6 + 4*x - 1)^6 +
    4*(12*(12*x^6 - 2*x + 1)^6 - 24*x^6 + 4*x - 1)^2
```

```
l(x) = f(g(h)); l
```

```
    x |--> 3*(12*(21*x^4 - 2*x^2 + 12)^6 - 42*x^4 + 4*x^2 - 23)^6 +
    4*(12*(21*x^4 - 2*x^2 + 12)^6 - 42*x^4 + 4*x^2 - 23)^2
```

```
fwerte = [f(x) for x in [-10..10]]; show(fwerte)
```

```
[3000400, 1594647, 786688, 353143, 140112, 46975, 12352, 2223, 2
```

```
# oder mit dictionaries:
fdict = {}
for x in [-10..10]:
   fdict[x] = f(x)
show(fdict)
```

```
{0 : 0, 1 : 7, 2 : 208, 3 : 2223, 4 : 12352, 5 : 46975, 6 : 140112, 7 : 3
```

```
gwerte = [g(x) for x in [-10..10]]; show(gwerte)
```

```
[12000021, 6377311, 3145745, 1411803, 559885, 187511, 49161, 87
```

```
kwerte = [k(x) for x in [-10..10]]; show(kwerte)
```

```
[63498131989070339447069614125374036271869011562703954
```

```
lwerte = [l(x) for x in [-10..10]]; show(lwerte)
```

```
[34522352423907595242974500248070710255892687780629600
```

```
filter(is_prime, gwerte)
```

```
[773, 11]
```

```
filter(is_prime, kwerte)
```

```
[]
```

Aufgabe 4

```
reset()
var('x')
```

```
x
```

```
f(x) = -x^2 + 2
g(x) = -4*sqrt(x) + 5
```

```
f(1), g(1)
```

```
(1, 1)
```

Aufgabe 5

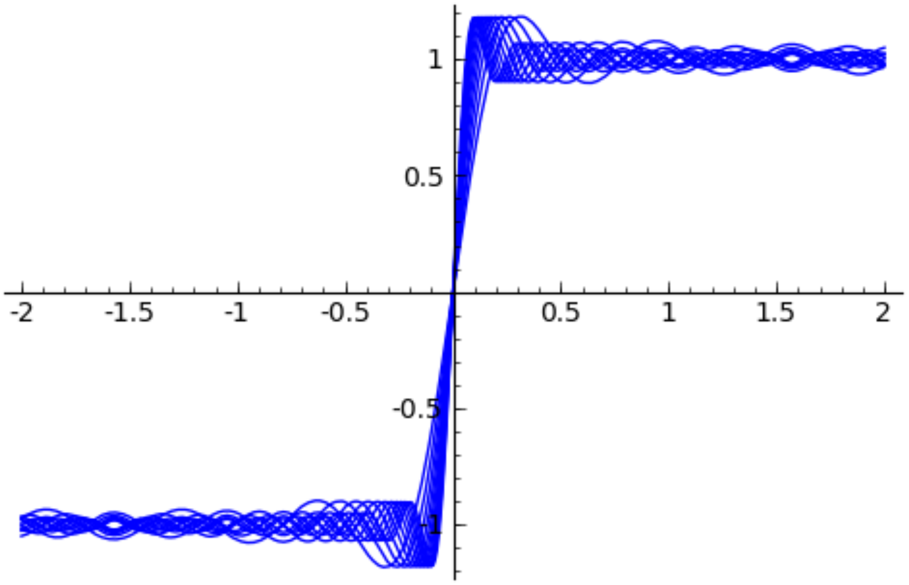
```
reset()
var('x,n,k')
```

```
    (x, n, k)
```

```
def s(k): return 4/pi * sum([ sin((2*n+1)*x)/(2*n+1) for n in [0..k] ])
f = [s(k) for k in [4..15]]; f
```

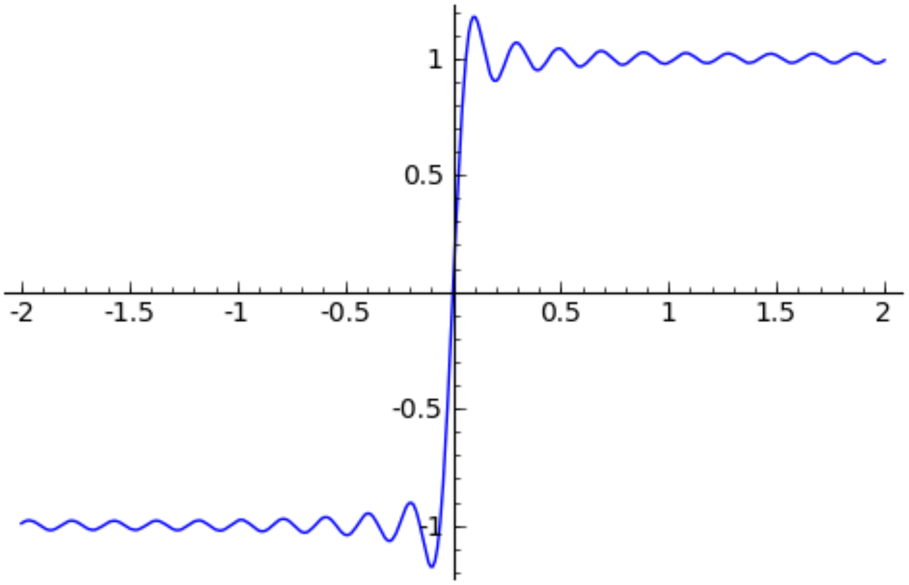
```
    [4/315*(105*sin(3*x) + 63*sin(5*x) + 45*sin(7*x) + 35*sin(9*x) +
    315*sin(x))/pi, 4/3465*(1155*sin(3*x) + 693*sin(5*x) + 495*sin(7*x)
    + 385*sin(9*x) + 315*sin(11*x) + 3465*sin(x))/pi,
    4/45045*(15015*sin(3*x) + 9009*sin(5*x) + 6435*sin(7*x) +
    5005*sin(9*x) + 4095*sin(11*x) + 3465*sin(13*x) + 45045*sin(x))/pi,
    4/45045*(15015*sin(3*x) + 9009*sin(5*x) + 6435*sin(7*x) +
    5005*sin(9*x) + 4095*sin(11*x) + 3465*sin(13*x) + 3003*sin(15*x) +
    45045*sin(x))/pi, 4/765765*(255255*sin(3*x) + 153153*sin(5*x) +
    109395*sin(7*x) + 85085*sin(9*x) + 69615*sin(11*x) + 58905*sin(13*x)
    + 51051*sin(15*x) + 45045*sin(17*x) + 765765*sin(x))/pi,
    4/14549535*(4849845*sin(3*x) + 2909907*sin(5*x) + 2078505*sin(7*x) +
    1616615*sin(9*x) + 1322685*sin(11*x) + 1119195*sin(13*x) +
    969969*sin(15*x) + 855855*sin(17*x) + 765765*sin(19*x) +
    14549535*sin(x))/pi, 4/14549535*(4849845*sin(3*x) + 2909907*sin(5*x)
    + 2078505*sin(7*x) + 1616615*sin(9*x) + 1322685*sin(11*x) +
    1119195*sin(13*x) + 969969*sin(15*x) + 855855*sin(17*x) +
    765765*sin(19*x) + 692835*sin(21*x) + 14549535*sin(x))/pi,
    4/334639305*(111546435*sin(3*x) + 66927861*sin(5*x) +
    47805615*sin(7*x) + 37182145*sin(9*x) + 30421755*sin(11*x) +
    25741485*sin(13*x) + 22309287*sin(15*x) + 19684665*sin(17*x) +
    17612595*sin(19*x) + 15935205*sin(21*x) + 14549535*sin(23*x) +
    334639305*sin(x))/pi, 4/1673196525*(557732175*sin(3*x) +
    334639305*sin(5*x) + 239028075*sin(7*x) + 185910725*sin(9*x) +
    152108775*sin(11*x) + 128707425*sin(13*x) + 111546435*sin(15*x) +
    98423325*sin(17*x) + 88062975*sin(19*x) + 79676025*sin(21*x) +
    72747675*sin(23*x) + 66927861*sin(25*x) + 1673196525*sin(x))/pi,
    4/5019589575*(1673196525*sin(3*x) + 1003917915*sin(5*x) +
    717084225*sin(7*x) + 557732175*sin(9*x) + 456326325*sin(11*x) +
    386122275*sin(13*x) + 334639305*sin(15*x) + 295269975*sin(17*x) +
    264188925*sin(19*x) + 239028075*sin(21*x) + 218243025*sin(23*x) +
    200783583*sin(25*x) + 185910725*sin(27*x) + 5019589575*sin(x))/pi,
    4/145568097675*(48522699225*sin(3*x) + 29113619535*sin(5*x) +
    20795442525*sin(7*x) + 16174233075*sin(9*x) + 13233463425*sin(11*x)
    + 11197545975*sin(13*x) + 9704539845*sin(15*x) +
    8562829275*sin(17*x) + 7661478825*sin(19*x) + 6931814175*sin(21*x) +
    6329047725*sin(23*x) + 5822723907*sin(25*x) + 5391411025*sin(27*x) +
    5019589575*sin(29*x) + 145568097675*sin(x))/pi,
    4/4512611027925*(1504203675975*sin(3*x) + 902522205585*sin(5*x) +
    644658718275*sin(7*x) + 501401225325*sin(9*x) +
    410237366175*sin(11*x) + 347123925225*sin(13*x) +
    300840735195*sin(15*x) + 265447707525*sin(17*x) +
    237505843575*sin(19*x) + 214886239425*sin(21*x) +
    196200479475*sin(23*x) + 180504441117*sin(25*x) +
    167133741775*sin(27*x) + 155607276825*sin(29*x) +
    145568097675*sin(31*x) + 4512611027925*sin(x))/pi]
```

```
plot(f, (x,-2,2))
```



```
a = animate([plot(s(k), (x,-2,2)) for k in [4..15]])
```

```
a.show(iterations=1)
```



Aufgabe 6

```
reset()
var('x')
```

```
    x
```

```
parametric_plot3d([5*cos(x), 2*sin(x), 4*x], (x,0,100), plot_points=200)
```

Aufgabe 7

```
reset()
def plot_dreiecke(l):
   ecken = Set(l)
   dreiecke = [s for s in ecken.subsets() if len(s)==3]
   p = Graphics()
   for d in dreiecke:
       p += polygon3d(d)
   p.show()
```

```
e1 = (1,0,0); e2 = (0,1,0); e3 = (0,0,1)
me1 = (-1,0,0); me2 = (0,-1,0); me3 = (0,0,-1)
plot_dreiecke([e1,e2,e3,me1,me2,me3])
```

```
octahedron()
```

Aufgabe 8

```
reset()
```

```
def f(v1,v2,w1,w2,w3):
   var('a,b')
   p = parametric_plot3d(v1+a*v2, (a,-5,5), thickness=3, color='green')
   p += parametric_plot3d(w1+a*w2+b*w3, (a,-5,5), (b,-4,4), opacity=0.8)
   p.show()
```

```
f(vector([0,0,0]),vector([1,1,0]),vector([1,0,0]),vector([0,1,1]),vector([1,0,1]))
```

Aufgabe 9

```
reset()
```

```
var('u,v')
```

```
    (u, v)
```

```
x(u,v) = cos(2*u)*cos(u+v)
y(u,v) = cos(2*u)*sin(u+v)
z(u,v) = sin(v)
```

```
parametric_plot3d([x,y,z], (u,-pi,pi), (v,-pi,pi), plot_points=[80,80])
```