lsg06

# Aufgabe 1

```
reset()
var('i,n')
```

```
(i, n)
```

```
limit( ((n+1)^2 - n^2)/n, n=oo )
```

```
2
```

```
limit( sqrt(n+1) - sqrt(n), n=oo )
```

```
0
```

```
limit( sum(i^99, i,1,n)/n^100, n=oo )
```

```
1/100
```

# Aufgabe 2

```
reset()
forget()
var('i')
```

```
i
```

```
sum(1/i^3, i,1,oo)
```

```
zeta(3)
```

```
# n(...) funktioniert nicht
zeta(3)
```

```
1.20205690315959
```

```
x=var('x'); assume(x>0); assume(x<1); sum(x^i, i,0,oo)
```

```
-1/(x - 1)
```

```
sum(1/(i+2*i^2), i,1,oo)
```

```
-2*log(2) + 2
```

# Aufgabe 3

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

```
(n, k)
```

```
a(n)=n^4-4*n^3
1/limit(a(n)^(1/n), n=oo), limit(a(n)/a(n+1), n=oo)
```

```
(1, 1)
```

```
a(n)=n^(log(n)/n)
1/limit(a(n)^(1/n), n=oo), limit(a(n)/a(n+1), n=oo)
```

```
(1, 1)
```

```
a(n)=sum(exp(k), k,0,n)
1/limit(a(n)^(1/n), n=oo), limit(a(n)/a(n+1), n=oo)
```

```
(e^(-1), e^(-1))
```

```
# NB: funktioniert nicht, wenn man e durch z.B. 2 ersetzt
limit( (2^(n+1)-1)^(1/n), n=oo ), limit( (e^(n+1)-1)^(1/n), n=oo )
```

```
(+Infinity, e)
```

```
# hier hilft die "taylor"-option:
limit( (2^(n+1)-1)^(1/n), n=oo, taylor=True )
```

```
2
```

# Aufgabe 4

```
reset()
```

```
# der rek-parameter zählt die rekursionstiefe und ist für die
# berechnung unötig
def a(n, rek):
   print " "*rek + "n = %i" % n
   if n==0 or n==1:
       return 1
   else:
       return a(n-2, rek+1) + a(n-1, rek+1)
a(3, 0)
```

```
n = 3
 n = 1
 n = 2
  n = 0
  n = 1
3
```

```
a=[1,1]
for n in [2..99]:
  a.append( a[n-1] + a[n-2] )
a
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393,
196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887,
9227465, 14930352, 24157817, 39088169, 63245986, 102334155,
165580141, 267914296, 433494437, 701408733, 1134903170, 1836311903,
2971215073, 4807526976, 7778742049, 12586269025, 20365011074,
32951280099, 53316291173, 86267571272, 139583862445, 225851433717,
365435296162, 591286729879, 956722026041, 1548008755920,
2504730781961, 4052739537881, 6557470319842, 10610209857723,
17167680177565, 27777890035288, 44945570212853, 72723460248141,
117669030460994, 190392490709135, 308061521170129, 498454011879264,
806515533049393, 1304969544928657, 2111485077978050,
3416454622906707, 5527939700884757, 8944394323791464,
14472334024676221, 23416728348467685, 37889062373143906,
61305790721611591, 99194853094755497, 160500643816367088,
259695496911122585, 420196140727489673, 679891637638612258,
1100087778366101931, 1779979416004714189, 2880067194370816120,
4660046610375530309, 7540113804746346429, 12200160415121876738,
19740274219868223167, 31940434634990099905, 51680708854858323072,
83621143489848422977, 135301852344706746049, 218922995834555169026,
354224848179261915075]
```

```
[(a[n]/a[n+1]).n(digits=10) for n in [0..98]]
```

```
[1.000000000, 0.5000000000, 0.6666666667, 0.6000000000,
0.6250000000, 0.6153846154, 0.6190476190, 0.6176470588,
0.6181818182, 0.6179775281, 0.6180555556, 0.6180257511,
0.6180371353, 0.6180327869, 0.6180344478, 0.6180338134,
0.6180340557, 0.6180339632, 0.6180339985, 0.6180339850,
0.6180339902, 0.6180339882, 0.6180339890, 0.6180339887,
0.6180339888, 0.6180339887, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
```

```
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888, 0.6180339888,
0.6180339888, 0.6180339888, 0.6180339888]
```

```
c = [( ((1+sqrt(5))/2)^n - ((1-sqrt(5))/2)^n )/sqrt(5) for n in [1..100]]
c = [x.n(digits=10) for x in c]
c
```

```
[1.000000000, 1.000000000, 2.000000000, 3.000000000, 5.000000000,
8.000000000, 13.00000000, 21.00000000, 34.00000000, 55.00000000,
89.00000000, 144.0000000, 233.0000000, 377.0000000, 610.0000000,
987.0000000, 1597.000000, 2584.000000, 4181.000000, 6765.000000,
10946.00000, 17711.00000, 28657.00000, 46368.00000, 75025.00000,
121393.0000, 196418.0000, 317811.0000, 514229.0000, 832040.0000,
1.346269000e6, 2.178309000e6, 3.524578000e6, 5.702887000e6,
9.227465000e6, 1.493035200e7, 2.415781700e7, 3.908816900e7,
6.324598600e7, 1.023341550e8, 1.655801410e8, 2.679142960e8,
4.334944370e8, 7.014087330e8, 1.134903170e9, 1.836311903e9,
2.971215073e9, 4.807526976e9, 7.778742049e9, 1.258626903e10,
2.036501107e10, 3.295128010e10, 5.331629118e10, 8.626757127e10,
1.395838624e11, 2.258514337e11, 3.654352962e11, 5.912867299e11,
9.567220261e11, 1.548008756e12, 2.504730782e12, 4.052739538e12,
6.557470320e12, 1.061020986e13, 1.716768018e13, 2.777789004e13,
4.494557021e13, 7.272346025e13, 1.176690305e14, 1.903924907e14,
3.080615212e14, 4.984540119e14, 8.065155331e14, 1.304969545e15,
2.111485078e15, 3.416454623e15, 5.527939701e15, 8.944394324e15,
1.447233403e16, 2.341672835e16, 3.788906237e16, 6.130579072e16,
9.919485310e16, 1.605006438e17, 2.596954969e17, 4.201961407e17,
6.798916377e17, 1.100087778e18, 1.779979416e18, 2.880067194e18,
4.660046611e18, 7.540113805e18, 1.220016042e19, 1.974027422e19,
3.194043464e19, 5.168070886e19, 8.362114349e19, 1.353018523e20,
2.189229958e20, 3.542248482e20]
```

```
[c[n]-a[n] for n in [0..99]]
```

```
[0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 0.0000000000, 0.0000000000, 2.328306437e-10,
4.656612873e-10, 4.656612873e-10, 9.313225746e-10, 0.0000000000,
3.725290298e-9, 0.0000000000, 7.450580597e-9, 7.450580597e-9,
0.0000000000, 2.980232239e-8, 5.960464478e-8, 5.960464478e-8,
1.192092896e-7, 2.384185791e-7, 2.384185791e-7, 9.536743164e-7,
9.536743164e-7, 1.907348633e-6, 5.722045898e-6, 3.814697266e-6,
0.00001144409180, 0.00001525878906, 0.00003051757812,
```

```
0.00003051757812, 0.00006103515625, 0.00006103515625,
0.0001220703125, 0.0003662109375, 0.0004882812500, 0.0009765625000,
0.001464843750, 0.001953125000, 0.005859375000, 0.003906250000,
0.01171875000, 0.01562500000, 0.01562500000, 0.04687500000,
0.06250000000, 0.1875000000, 0.1250000000, 0.3750000000,
0.5000000000, 1.000000000, 2.000000000, 2.000000000, 4.000000000,
6.000000000, 12.00000000, 16.00000000, 32.00000000, 48.00000000,
96.00000000, 128.0000000, 128.0000000, 384.0000000, 384.0000000,
1024.000000, 1024.000000, 2048.000000, 3072.000000, 6144.000000,
8192.000000, 16384.00000, 32768.00000, 49152.00000, 81920.00000,
131072.0000, 196608.0000, 262144.0000, 524288.0000, 786432.0000,
1.572864000e6, 2.621440000e6, 4.194304000e6, 6.291456000e6,
1.048576000e7, 1.677721600e7, 3.355443200e7, 2.516582400e7,
8.388608000e7, 1.006632960e8, 2.013265920e8, 4.026531840e8,
5.368709120e8, 8.053063680e8, 1.342177280e9, 2.147483648e9,
4.294967296e9, 4.294967296e9, 8.589934592e9, 1.288490189e10]
```

```
[c[n]/a[n] for n in [0..99]]
```

```
[1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000,
1.000000000, 1.000000000, 1.000000000, 1.000000000, 1.000000000]
```

```
c[99]/c[98]
```

```
1.618033989
```

```
# analytisch (für die ersten 10 elemenente)
c = [( ((1+sqrt(5))/2)^n - ((1-sqrt(5))/2)^n )/sqrt(5) for n in [1..10]]
map(expand, c)
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

# Aufgabe 5

```
reset()
```

```
a = {}
for n in [1..10]:
   a[n] = log(n)
a
```

```
{1: 0, 2: log(2), 3: log(3), 4: log(4), 5: log(5), 6: log(6), 7:
log(7), 8: log(8), 9: log(9), 10: log(10)}
```

```
a[11] = log(11); a
```

```
{1: 0, 2: log(2), 3: log(3), 4: log(4), 5: log(5), 6: log(6), 7:
log(7), 8: log(8), 9: log(9), 10: log(10), 11: log(11)}
```

```
k = [x for x in a]
k.sort()
k
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
v = [a[x] for x in a]
v.sort()
v
```

```
[0, log(2), log(3), log(4), log(5), log(6), log(7), log(8), log(9),
log(10), log(11)]
```

# Aufgabe 6

```
reset()
var('a,x')
```

```
(a, x)
```

```
forget(); assume(a>0); limit(x^a, x=oo)
```

```
+Infinity
```

```
forget(); assume(a==0); limit(x^a, x=oo)
```

```
1
```

```
forget(); assume(a<0); limit(x^a, x=oo)
```

```
0
```

# Aufgabe 7

```
reset();forget()
var('n')
```

```
n
```

```
1/limit( (1/(2*n+1))^(1/(2*n+1)), n=oo )
```

```
1
```

```
assume(abs(x)<1, x<>0)
f(x) = sum((-1)^n * x^(2*n+1)/(2*n+1), n,0,oo); f
```

```
x |--> arctan(x)
```

```
4*(4*f(1/5)-f(1/239))
```

```
-4*arctan(1/239) + 16*arctan(1/5)
```

```
def f(x): return sum((-1)^n * x^(2*n+1)/(2*n+1), n,0,oo)
```

```
4*(4*f(1/5)-f(1/239))
```

```
-4*arctan(1/239) + 16*arctan(1/5)
```

```
( 4*(4*f(1/5)-f(1/239)) ).n()
```

```
3.14159265358979
```

# Aufgabe 8

```
reset()
var('a,b,c,d,x')
g(x) = a*x^3 + b*x^2 + c*x + d
```

```
eq1 = g(-2) == 3
eq2 = g.diff(x)(-2) == 0
```

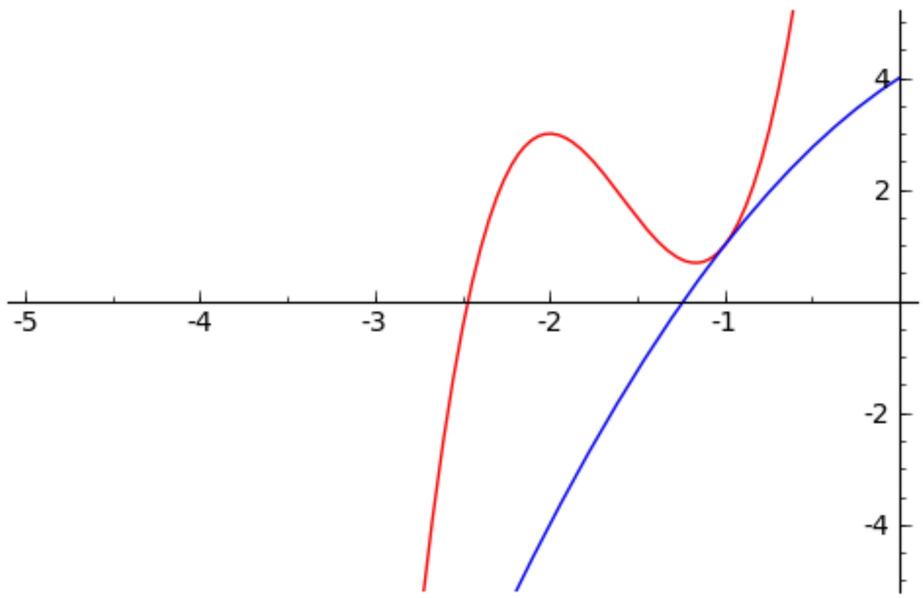
```
f(x) = -x^2 + 2*x + 4
eq3 = g(-1) == f(-1)
```

```
eq4 = g.diff(x)(-1) == f.diff(x)(-1)

s=solve([eq1,eq2,eq3,eq4], [a,b,c,d], solution_dict=True); s
```

```
[{d: 27, c: 56, b: 38, a: 8}]
```

```
p = plot(g.subs(s[0])(x), x,-3,0, color='red')
p += plot(f(x), x,-5,0)
p.show(ymin=-5,ymax=5)
```




## Aufgabe 9

```
reset()

# 1. Möglichkeit
def f(l):
   ret = {}
   # Set(l) zum Entfernen doppelter Elemente
   for x in Set(l):
       ret[x] = len([s for s in l if s==x])
   return ret
f([1,1,2,3,2,4,3,3,1])
```

```
{1: 3, 2: 2, 3: 3, 4: 1}
```

```
# 2. Möglichkeit
def f(l):
```

```
    ret = {}
    for x in l:
        if x in ret:
            ret[x] = ret[x] + 1
        else:
            ret[x] = 1
    return ret
f([1,1,2,3,2,4,3,3,1])
```

```
{1: 3, 2: 2, 3: 3, 4: 1}
```

# Aufgabe 10

```
reset()
```

```
def newton(f, x0, n):
    x = [x0.n()]
    df = f.diff()
    for k in [0..n-1]:
        x.append(x[k] - f(x[k])/df(x[k]))
    return x
```

```
f(x) = exp(-x)-x
n = newton(f, 0, 20); n
```

```
[0.000000000000000, 0.500000000000000, 0.566311003197218,
0.567143165034862, 0.567143290409781, 0.567143290409784,
0.567143290409784, 0.567143290409784, 0.567143290409784,
0.567143290409784, 0.567143290409784, 0.567143290409784,
0.567143290409784, 0.567143290409784, 0.567143290409784,
0.567143290409784, 0.567143290409784, 0.567143290409784,
0.567143290409784, 0.567143290409784, 0.567143290409784]
```

```
map(f, n)
```

```
[1.00000000000000, 0.106530659712633, 0.00130450980602004,
1.96480471781335e-7, 4.44089209850063e-15, -1.11022302462516e-16,
0.000000000000000, 0.000000000000000, 0.000000000000000,
0.000000000000000, 0.000000000000000, 0.000000000000000,
0.000000000000000, 0.000000000000000, 0.000000000000000,
0.000000000000000, 0.000000000000000, 0.000000000000000,
0.000000000000000, 0.000000000000000, 0.000000000000000]
```

```
g(x) = x^3 - 2*x + 3
n = newton(g, 0, 20); n
```

```
[0.000000000000000, 1.50000000000000, 0.789473684210526,
15.4837625979843, 10.3471096468676, 6.93189687901041,
4.66517856298333, 3.16099837457084, 2.15075075586611,
1.42269211389378, 0.677578739255802, 3.81881803775170,
2.59598052875135, 1.75597611228197, 1.07980023875699,
-0.321765089204953, 1.81521452010056, 1.13662200576445,
-0.0336798562672256, 1.50259486794423, 0.792959307842975]
```

```
map(g, n)
```

```
[3.00000000000000, 3.37500000000000, 1.91310686689022,
3684.21662289513, 1090.09504951794, 322.222130328704,
95.2020807287257, 28.2624168084041, 8.64728823110586,
3.03421980134267, 1.95592769389672, 51.0536049351272,
15.3026500185986, 4.90251601852656, 2.09941264722727,
3.61021694658863, 5.35070961781631, 2.19516885108987,
3.06732150837154, 3.38735594023562, 1.91268187758649]
```

```
plot(g(x), x,-3,3)
```

```
newton(g, -4, 20)
```

```
[-4.00000000000000, -2.84782608695652, -2.20293934427889,
-1.94138162935686, -1.89472462010931, -1.89329053142704,
-1.89328919630565, -1.89328919630450, -1.89328919630450,
-1.89328919630450, -1.89328919630450, -1.89328919630450,
-1.89328919630450, -1.89328919630450, -1.89328919630450,
-1.89328919630450, -1.89328919630450, -1.89328919630450,
-1.89328919630450, -1.89328919630450, -1.89328919630450]
```

```
def newton_rekursiv(f, x0, n):
    if n==0:
        return x0.n()
    else:
        y = newton_rekursiv(f, x0, n-1)
        return y - f(y)/f.diff()(y)
```

```
f(x) = exp(-x)-x
n = newton_rekursiv(f, 0, 20); n
```

```
0.567143290409784
```